Introduction to finite element methods

Hans Petter Langtangen[1,2]

[1]Center for Biomedical Computing, Simula Research Laboratory
[2]Department of Informatics, University of Oslo

Sep 25, 2014

PRELIMINARY VERSION

Contents

Approximation of vectors **6**
1.1 Approximation of planar vectors . . . 6
1.2 Approximation of general vectors . . . 9

Approximation of functions **12**
2.1 The least squares method . . . 12
2.2 The projection (or Galerkin) method . . . 13
2.3 Example: linear approximation . . . 14
2.4 Implementation of the least squares method . . . 15
2.5 Perfect approximation . . . 17
2.6 Ill-conditioning . . . 18
2.7 Fourier series . . . 20
2.8 Orthogonal basis functions . . . 21
2.9 Numerical computations . . . 23
2.10 The interpolation (or collocation) method . . . 24
2.11 Lagrange polynomials . . . 26

Finite element basis functions **32**
3.1 Elements and nodes . . . 33
3.2 The basis functions . . . 36
3.3 Example on piecewise quadratic finite element functions . . . 37
3.4 Example on piecewise linear finite element functions . . . 39
3.5 Example on piecewise cubic finite element basis functions . . . 40
3.6 Calculating the linear system . . . 41
3.7 Assembly of elementwise computations . . . 44
3.8 Mapping to a reference element . . . 46
3.9 Example: Integration over a reference element . . . 49

4 Implementation
4.1 Integration . . .
4.2 Linear system assembly and solution . . .
4.3 Example on computing symbolic approximations . . .
4.4 Comparison with finite elements and interpolation/collocation
4.5 Example on computing numerical approximations . . .
4.6 The structure of the coefficient matrix . . .
4.7 Applications . . .
4.8 Sparse matrix storage and solution . . .

5 Comparison of finite element and finite difference approxi tion
5.1 Finite difference approximation of given functions . . .
5.2 Finite difference interpretation of a finite element approximati
5.3 Making finite elements behave as finite differences . . .

6 A generalized element concept
6.1 Cells, vertices, and degrees of freedom . . .
6.2 Extended finite element concept . . .
6.3 Implementation . . .
6.4 Computing the error of the approximation . . .
6.5 Example: Cubic Hermite polynomials . . .

7 Numerical integration
7.1 Newton-Cotes rules . . .
7.2 Gauss-Legendre rules with optimized points . . .

8 Approximation of functions in 2D
8.1 2D basis functions as tensor products of 1D functions . . .
8.2 Example: Polynomial basis in 2D . . .
8.3 Implementation . . .
8.4 Extension to 3D . . .

9 Finite elements in 2D and 3D
9.1 Basis functions over triangles in the physical domain . . .
9.2 Basis functions over triangles in the reference cell . . .
9.3 Affine mapping of the reference cell . . .
9.4 Isoparametric mapping of the reference cell . . .
9.5 Computing integrals . . .

10 Exercises

11 Basic principles for approximating differential equations
11.1 Differential equation models . . .
11.2 Simple model problems . . .
11.3 Forming the residual . . .
11.4 The least squares method . . .

11.5 The Galerkin method . . . 96
11.6 The Method of Weighted Residuals . . . 96
11.7 Test and Trial Functions . . . 97
11.8 The collocation method . . . 97
11.9 Examples on using the principles . . . 99
11.10 Integration by parts . . . 102
11.11 Boundary function . . . 103
11.12 Abstract notation for variational formulations . . . 104
11.13 Variational problems and optimization of functionals . . . 106

2 Examples on variational formulations 106
12.1 Variable coefficient . . . 107
12.2 First-order derivative in the equation and boundary condition . . . 108
12.3 Nonlinear coefficient . . . 110
12.4 Computing with Dirichlet and Neumann conditions . . . 111
12.5 When the numerical method is exact . . . 112

3 Computing with finite elements 112
13.1 Finite element mesh and basis functions . . . 112
13.2 Computation in the global physical domain . . . 113
13.3 Comparison with a finite difference discretization . . . 115
13.4 Cellwise computations . . . 116

4 Boundary conditions: specified nonzero value 119
14.1 General construction of a boundary function . . . 119
14.2 Example on computing with finite element-based a boundary function . . . 121
14.3 Modification of the linear system . . . 122
14.4 Symmetric modification of the linear system . . . 125
14.5 Modification of the element matrix and vector . . . 126

5 Boundary conditions: specified derivative 127
15.1 The variational formulation . . . 127
15.2 Boundary term vanishes because of the test functions . . . 127
15.3 Boundary term vanishes because of linear system modifications . . . 128
15.4 Direct computation of the global linear system . . . 128
15.5 Cellwise computations . . . 130

6 Implementation 131
16.1 Global basis functions . . . 131
16.2 Example: constant right-hand side . . . 133
16.3 Finite elements . . . 134

7 Variational formulations in 2D and 3D 136
17.1 Transformation to a reference cell in 2D and 3D . . . 138
17.2 Numerical integration . . . 139
17.3 Convenient formulas for P1 elements in 2D . . . 140

18 Summary

19 Time-dependent problems
19.1 Discretization in time by a Forward Euler scheme . . .
19.2 Variational forms . . .
19.3 Simplified notation for the solution at recent time levels . . .
19.4 Deriving the linear systems . . .
19.5 Computational algorithm . . .
19.6 Comparing P1 elements with the finite difference method . . .
19.7 Discretization in time by a Backward Euler scheme . . .
19.8 Dirichlet boundary conditions . . .
19.9 Example: Oscillating Dirichlet boundary condition . . .
19.10 Analysis of the discrete equations . . .

20 Systems of differential equations
20.1 Variational forms . . .
20.2 A worked example . . .
20.3 Identical function spaces for the unknowns . . .
20.4 Different function spaces for the unknowns . . .
20.5 Computations in 1D . . .

21 Exercises

The finite element method is a powerful tool for solving differential eq
The method can easily deal with complex geometries and higher-order app
tions of the solution. Figure 1 shows a two-dimensional domain with a no
geometry. The idea is to divide the domain into triangles (elements) a
a polynomial approximations to the unknown functions on each triang
method glues these piecewise approximations together to find a global s
Linear and quadratic polynomials over the triangles are particularly po

Many successful numerical methods for differential equations, includ
finite element method, aim at approximating the unknown function by

$$u(x) = \sum_{i=0}^{N} c_i \psi_i(x),$$

where $\psi_i(x)$ are prescribed functions and $c_0, \ldots, c_N$ are unknown coeffic
be determined. Solution methods for differential equations utilizing (
have a *principle* for constructing $N+1$ equations to determine $c_0, \ldots, c_I$
there is a *machinery* regarding the actual constructions of the equat
$c_0, \ldots, c_N$, in a particular problem. Finally, there is a *solve* phase for co
the solution $c_0, \ldots, c_N$ of the $N+1$ equations.

Especially in the finite element method, the machinery for construc
discrete equations to be implemented on a computer is quite comprehensi
many mathematical and implementational details entering the scene at t

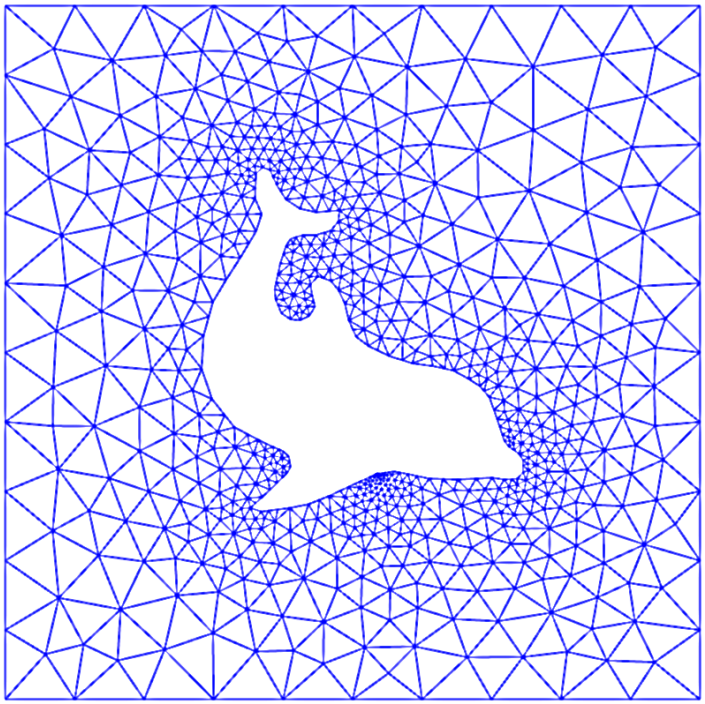


Figure 1: Domain for flow around a dolphin.

me. From an ease-of-learning perspective it can therefore be wise to follow n idea of Larson and Bengzon [1] and introduce the computational machinery or a trivial equation: $u = f$. Solving this equation with f given and u on the orm (1) means that we seek an approximation u to f. This approximation roblem has the advantage of introducing most of the finite element toolbox, ut with postponing demanding topics related to differential equations (e.g., ntegration by parts, boundary conditions, and coordinate mappings). This is ne reason why we shall first become familiar with finite element *approximation* efore addressing finite element methods for differential equations.

First, we refresh some linear algebra concepts about approximating vectors n vector spaces. Second, we extend these concepts to approximating functions n function spaces, using the same principles and the same notation. We present xamples on approximating functions by global basis functions with support roughout the entire domain. Third, we introduce the finite element type of ocal basis functions and explain the computational algorithms for working with uch functions. Three types of approximation principles are covered: 1) the least quares method, 2) the L_2 projection or Galerkin method, and 3) interpolation r collocation.

1 Approximation of vectors

We shall start with introducing two fundamental methods for determi coefficients c_i in (1) and illustrate the methods on approximation of because vectors in vector spaces give a more intuitive understanding than directly with approximation of functions in function spaces. The ex from vectors to functions will be trivial as soon as the fundamental ic understood.

The first method of approximation is called the *least squares met* consists in finding c_i such that the difference $u - f$, measured in some minimized. That is, we aim at finding the best approximation u to f (norm). The second method is not as intuitive: we find u such that t $u - f$ is orthogonal to the space where we seek u. This is known as *pr* or we may also call it a *Galerkin method*. When approximating vect functions, the two methods are equivalent, but this is no longer the ca applying the principles to differential equations.

1.1 Approximation of planar vectors

Suppose we have given a vector $\boldsymbol{f} = (3, 5)$ in the xy plane and that we approximate this vector by a vector aligned in the direction of the vect Figure 2 depicts the situation.

We introduce the vector space V spanned by the vector $\boldsymbol{\psi}_0 = (a, b)$:

$$V = \text{span}\,\{\boldsymbol{\psi}_0\}\,.$$

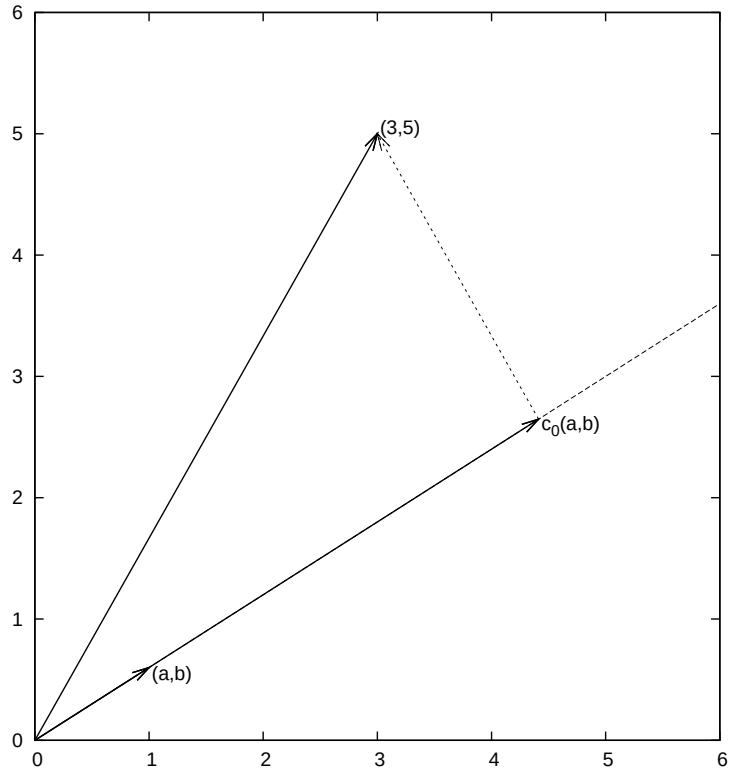
We say that $\boldsymbol{\psi}_0$ is a basis vector in the space V. Our aim is to f vector $\boldsymbol{u} = c_0\boldsymbol{\psi}_0 \in V$ which best approximates the given vector $\boldsymbol{f} = ($ reasonable criterion for a best approximation could be to minimize the l the difference between the approximate $\boldsymbol{u}$ and the given $\boldsymbol{f}$. The differ error $\boldsymbol{e} = \boldsymbol{f} - \boldsymbol{u}$, has its length given by the *norm*

$$||\boldsymbol{e}|| = (\boldsymbol{e}, \boldsymbol{e})^{\frac{1}{2}},$$

where $(\boldsymbol{e}, \boldsymbol{e})$ is the *inner product* of $\boldsymbol{e}$ and itself. The inner product, als *scalar product* or *dot product*, of two vectors $\boldsymbol{u} = (u_0, u_1)$ and $\boldsymbol{v} = (v$ defined as

$$(\boldsymbol{u}, \boldsymbol{v}) = u_0v_0 + u_1v_1\,.$$

Remark 1. We should point out that we use the notation $(\cdot, \cdot)$ for two things: (a, b) for scalar quantities a and b means the vector starting in th and ending in the point (a, b), while $(\boldsymbol{u}, \boldsymbol{v})$ with vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ means t product of these vectors. Since vectors are here written in boldface fo should be no confusion. We may add that the norm associated with th product is the usual Eucledian length of a vector.



igure 2: Approximation of a two-dimensional vector by a one-dimensional ector.

emark 2. It might be wise to refresh some basic linear algebra by consulting textbook. Exercises 1 and 2 suggest specific tasks to regain familiarity with ndamental operations on inner product vector spaces.

he least squares method. We now want to find c_0 such that it minimizes $\boldsymbol{e}||$. The algebra is simplified if we minimize the square of the norm, $||\boldsymbol{e}||^2 = \cdot, \boldsymbol{e})$, instead of the norm itself. Define the function

$$E(c_0) = (\boldsymbol{e}, \boldsymbol{e}) = (\boldsymbol{f} - c_0\boldsymbol{\psi}_0, \boldsymbol{f} - c_0\boldsymbol{\psi}_0)\,. \tag{4}$$

/e can rewrite the expressions of the right-hand side in a more convenient form or further work:

$$E(c_0) = (\boldsymbol{f}, \boldsymbol{f}) - 2c_0(\boldsymbol{f}, \boldsymbol{\psi}_0) + c_0^2(\boldsymbol{\psi}_0, \boldsymbol{\psi}_0)\,. \tag{5}$$

he rewrite results from using the following fundamental rules for inner product paces:

$$(\alpha\boldsymbol{u}, \boldsymbol{v}) = \alpha(\boldsymbol{u}, \boldsymbol{v}), \quad \alpha \in \mathbb{R},$$

$$(\boldsymbol{u} + \boldsymbol{v}, \boldsymbol{w}) = (\boldsymbol{u}, \boldsymbol{w}) + (\boldsymbol{v}, \boldsymbol{w}),$$

$$(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{v}, \boldsymbol{u})\,.$$

Minimizing $E(c_0)$ implies finding c_0 such that

$$\frac{\partial E}{\partial c_0} = 0\,.$$

Differentiating (5) with respect to c_0 gives

$$\frac{\partial E}{\partial c_0} = -2(\boldsymbol{f}, \boldsymbol{\psi}_0) + 2c_0(\boldsymbol{\psi}_0, \boldsymbol{\psi}_0)\,.$$

Setting the above expression equal to zero and solving for c_0 gives

$$c_0 = \frac{(\boldsymbol{f}, \boldsymbol{\psi}_0)}{(\boldsymbol{\psi}_0, \boldsymbol{\psi}_0)},$$

which in the present case with $\boldsymbol{\psi}_0 = (a, b)$ results in

$$c_0 = \frac{3a + 5b}{a^2 + b^2}\,.$$

For later, it is worth mentioning that setting the key equation (9) can be rewritten as

$$(\boldsymbol{f} - c0\boldsymbol{\psi}_0, \boldsymbol{\psi}_0) = 0,$$

or

$$(\boldsymbol{e}, \boldsymbol{\psi}_0) = 0\,.$$

The projection method. We shall now show that minimizing $||\boldsymbol{e}||^2$ that $\boldsymbol{e}$ is orthogonal to *any* vector $\boldsymbol{v}$ in the space V. This result is visual clear from Figure 2 (think of other vectors along the line (a, b): all of th lead to a larger distance between the approximation and $\boldsymbol{f}$). To see th mathematically, we express any $\boldsymbol{v} \in V$ as $\boldsymbol{v} = s\boldsymbol{\psi}_0$ for any scalar para recall that two vectors are orthogonal when their inner product vanish

alculate the inner product

$$\begin{aligned}(\boldsymbol{e}, s\boldsymbol{\psi}_0) &= (\boldsymbol{f} - c_0\boldsymbol{\psi}_0, s\boldsymbol{\psi}_0)\\ &= (\boldsymbol{f}, s\boldsymbol{\psi}_0) - (c_0\boldsymbol{\psi}_0, s\boldsymbol{\psi}_0)\\ &= s(\boldsymbol{f}, \boldsymbol{\psi}_0) - sc_0(\boldsymbol{\psi}_0, \boldsymbol{\psi}_0)\\ &= s(\boldsymbol{f}, \boldsymbol{\psi}_0) - s\frac{(\boldsymbol{f}, \boldsymbol{\psi}_0)}{(\boldsymbol{\psi}_0, \boldsymbol{\psi}_0)}(\boldsymbol{\psi}_0, \boldsymbol{\psi}_0)\\ &= s\left((\boldsymbol{f}, \boldsymbol{\psi}_0) - (\boldsymbol{f}, \boldsymbol{\psi}_0)\right)\\ &= 0\,.\end{aligned}$$

herefore, instead of minimizing the square of the norm, we could demand that is orthogonal to any vector in V. This method is known as *projection*, because is the same as projecting the vector onto the subspace. (The approach can lso be referred to as a Galerkin method as explained at the end of Section 1.2.)

Mathematically the projection method is stated by the equation

$$(\boldsymbol{e}, \boldsymbol{v}) = 0, \quad \forall \boldsymbol{v} \in V\,. \tag{13}$$

n arbitrary $\boldsymbol{v} \in V$ can be expressed as $s\boldsymbol{\psi}_0$, $s \in \mathbb{R}$, and therefore (13) implies

$$(\boldsymbol{e}, s\boldsymbol{\psi}_0) = s(\boldsymbol{e}, \boldsymbol{\psi}_0) = 0,$$

hich means that the error must be orthogonal to the basis vector in the space :

$$(\boldsymbol{e}, \boldsymbol{\psi}_0) = 0 \quad \text{or} \quad (\boldsymbol{f} - c_0\boldsymbol{\psi}_0, \boldsymbol{\psi}_0) = 0\,.$$

he latter equation gives (10) and it also arose from least squares computations ı (12).

.2 Approximation of general vectors

et us generalize the vector approximation from the previous section to vectors ı spaces with arbitrary dimension. Given some vector $\boldsymbol{f}$, we want to find the est approximation to this vector in the space

$$V = \text{span}\,\{\boldsymbol{\psi}_0, \ldots, \boldsymbol{\psi}_N\}\,.$$

Ve assume that the *basis vectors* $\boldsymbol{\psi}_0, \ldots, \boldsymbol{\psi}_N$ are linearly independent so that one of them are redundant and the space has dimension $N + 1$. Any vector $\in V$ can be written as a linear combination of the basis vectors,

$$\boldsymbol{u} = \sum_{j=0}^{N} c_j\boldsymbol{\psi}_j,$$

here $c_j \in \mathbb{R}$ are scalar coefficients to be determined.

The least squares method. Now we want to find $c_0, \ldots, c_N$, such t the best approximation to $\boldsymbol{f}$ in the sense that the distance (error) $\boldsymbol{e}$ = is minimized. Again, we define the squared distance as a function of parameters $c_0, \ldots, c_N$,

$$\begin{aligned}E(c_0, \ldots, c_N) = (\boldsymbol{e}, \boldsymbol{e}) &= (\boldsymbol{f} - \sum_j c_j\boldsymbol{\psi}_j, \boldsymbol{f} - \sum_j c_j\boldsymbol{\psi}_j)\\ &= (\boldsymbol{f}, \boldsymbol{f}) - 2\sum_{j=0}^{N} c_j(\boldsymbol{f}, \boldsymbol{\psi}_j) + \sum_{p=0}^{N}\sum_{q=0}^{N} c_pc_q(\boldsymbol{\psi}_p, \boldsymbol{\psi}_q)\,.\end{aligned}$$

Minimizing this E with respect to the independent variables $c_0, \ldots, c_N$ is c by requiring

$$\frac{\partial E}{\partial c_i} = 0, \quad i = 0, \ldots, N\,.$$

The second term in (14) is differentiated as follows:

$$\frac{\partial}{\partial c_i}\sum_{j=0}^{N} c_j(\boldsymbol{f}, \boldsymbol{\psi}_j) = (\boldsymbol{f}, \boldsymbol{\psi}_i),$$

since the expression to be differentiated is a sum and only one term, c contains c_i and this term is linear in c_i. To understand this differenti detail, write out the sum specifically for, e.g, $N = 3$ and $i = 1$.

The last term in (14) is more tedious to differentiate. We start with

$$\frac{\partial}{\partial c_i}c_pc_q = \begin{cases} 0, & \text{if } p \neq i \text{ and } q \neq i,\\ c_q, & \text{if } p = i \text{ and } q \neq i,\\ c_p, & \text{if } p \neq i \text{ and } q = i,\\ 2c_i, & \text{if } p = q = i,\end{cases}$$

Then

$$\frac{\partial}{\partial c_i}\sum_{p=0}^{N}\sum_{q=0}^{N} c_pc_q(\boldsymbol{\psi}_p, \boldsymbol{\psi}_q) = \sum_{p=0, p\neq i}^{N} c_p(\boldsymbol{\psi}_p, \boldsymbol{\psi}_i) + \sum_{q=0, q\neq i}^{N} c_q(\boldsymbol{\psi}_q, \boldsymbol{\psi}_i) + 2c_i($$

The last term can be included in the other two sums, resulting in

$$\frac{\partial}{\partial c_i}\sum_{p=0}^{N}\sum_{q=0}^{N} c_pc_q(\boldsymbol{\psi}_p, \boldsymbol{\psi}_q) = 2\sum_{j=0}^{N} c_i(\boldsymbol{\psi}_j, \boldsymbol{\psi}_i)\,.$$

It then follows that setting

$$\frac{\partial E}{\partial c_i} = 0, \quad i = 0, \ldots, N,$$

ads to a linear system for $c_0, \ldots, c_N$:

$$\sum_{j=0}^{N} A_{i,j} c_j = b_i, \quad i = 0, \ldots, N, \tag{18}$$

here

$$A_{i,j} = (\boldsymbol{\psi}_i, \boldsymbol{\psi}_j), \tag{19}$$
$$b_i = (\boldsymbol{\psi}_i, \boldsymbol{f})\,. \tag{20}$$

/e have changed the order of the two vectors in the inner product according to .1):

$$A_{i,j} = (\boldsymbol{\psi}_j, \boldsymbol{\psi}_i) = (\boldsymbol{\psi}_i, \boldsymbol{\psi}_j),$$

mply because the sequence *i-j* looks more aesthetic.

'he Galerkin or projection method. In analogy with the "one-dimensional" xample in Section 1.1, it holds also here in the general case that minimizing the istance (error) $\boldsymbol{e}$ is equivalent to demanding that $\boldsymbol{e}$ is orthogonal to all $\boldsymbol{v} \in V$:

$$(\boldsymbol{e}, \boldsymbol{v}) = 0, \quad \forall \boldsymbol{v} \in V\,. \tag{21}$$

ince any $\boldsymbol{v} \in V$ can be written as $\boldsymbol{v} = \sum_{i=0}^{N} c_i \boldsymbol{\psi}_i$, the statement (21) is quivalent to saying that

$$(\boldsymbol{e}, \sum_{i=0}^{N} c_i \boldsymbol{\psi}_i) = 0,$$

or any choice of coefficients $c_0, \ldots, c_N$. The latter equation can be rewritten as

$$\sum_{i=0}^{N} c_i (\boldsymbol{e}, \boldsymbol{\psi}_i) = 0\,.$$

this is to hold for arbitrary values of $c_0, \ldots, c_N$ we must require that each erm in the sum vanishes,

$$(\boldsymbol{e}, \boldsymbol{\psi}_i) = 0, \quad i = 0, \ldots, N\,. \tag{22}$$

hese $N+1$ equations result in the same linear system as (18):

$$(\boldsymbol{f} - \sum_{j=0}^{N} c_j \boldsymbol{\psi}_j, \boldsymbol{\psi}_i) = (\boldsymbol{f}, \boldsymbol{\psi}_i) - \sum_{j \in \mathcal{I}_s} (\boldsymbol{\psi}_i, \boldsymbol{\psi}_j) c_j = 0,$$

nd hence

$$\sum_{j=0}^{N} (\boldsymbol{\psi}_i, \boldsymbol{\psi}_j) c_j = (\boldsymbol{f}, \boldsymbol{\psi}_i), \quad i = 0, \ldots, N\,.$$

So, instead of differentiating the $E(c_0, \ldots, c_N)$ function, we could sim (21) as the principle for determining $c_0, \ldots, c_N$, resulting in the $N+1$ e (22).

The names *least squares method* or *least squares approximation* are since the calculations consists of minimizing $||\boldsymbol{e}||^2$, and $||\boldsymbol{e}||^2$ is a sum of of differences between the components in $\boldsymbol{f}$ and $\boldsymbol{u}$. We find $\boldsymbol{u}$ such t sum of squares is minimized.

The principle (21), or the equivalent form (22), is known as *pr* Almost the same mathematical idea was used by the Russian mathen Boris Galerkin[1] to solve differential equations, resulting in what is widel as *Galerkin's method*.

2 Approximation of functions

Let V be a function space spanned by a set of *basis functions* $\psi_0, \ldots, \psi$

$$V = \text{span}\,\{\psi_0, \ldots, \psi_N\},$$

such that any function $u \in V$ can be written as a linear combination of t functions:

$$u = \sum_{j \in \mathcal{I}_s} c_j \psi_j\,.$$

The index set $\mathcal{I}_s$ is defined as $\mathcal{I}_s = \{0, \ldots, N\}$ and is used both for c notation and for flexibility in the numbering of elements in sequences.

For now, in this introduction, we shall look at functions of a single x: $u = u(x)$, $\psi_i = \psi_i(x)$, $i \in \mathcal{I}_s$. Later, we will almost trivially ext mathematical details to functions of two- or three-dimensional physica The approximation (23) is typically used to discretize a problem in spac methods, most notably finite differences, are common for time discre although the form (23) can be used in time as well.

2.1 The least squares method

Given a function $f(x)$, how can we determine its best approximation $u($ A natural starting point is to apply the same reasoning as we did for in Section 1.2. That is, we minimize the distance between u and f. E this requires a norm for measuring distances, and a norm is most conv defined through an inner product. Viewing a function as a vector of i many point values, one for each value of x, the inner product could int

[1] `http://en.wikipedia.org/wiki/Boris_Galerkin`

e defined as the usual summation of pairwise components, with summation eplaced by integration:

$$(f, g) = \int f(x)g(x)\ \mathrm{d}x\,.$$

o fix the integration domain, we let $f(x)$ and $\psi_i(x)$ be defined for a domain $\subset \mathbb{R}$. The inner product of two functions $f(x)$ and $g(x)$ is then

$$(f, g) = \int_\Omega f(x)g(x)\ \mathrm{d}x\,. \tag{24}$$

The distance between f and any function $u \in V$ is simply $f - u$, and the quared norm of this distance is

$$E = (f(x) - \sum_{j\in\mathcal{I}_s} c_j\psi_j(x), f(x) - \sum_{j\in\mathcal{I}_s} c_j\psi_j(x))\,. \tag{25}$$

ote the analogy with (14): the given function f plays the role of the given ector $\boldsymbol{f}$, and the basis function ψ_i plays the role of the basis vector $\boldsymbol{\psi}_i$. We can ewrite (25), through similar steps as used for the result (14), leading to

$$E(c_i, \ldots, c_N) = (f, f) - 2\sum_{j\in\mathcal{I}_s} c_j(f, \psi_i) + \sum_{p\in\mathcal{I}_s}\sum_{q\in\mathcal{I}_s} c_p c_q(\psi_p, \psi_q)\,. \tag{26}$$

Minimizing this function of $N+1$ scalar variables $\{c_i\}_{i\in\mathcal{I}_s}$, requires differentiation ith respect to c_i, for all $i \in \mathcal{I}_s$. The resulting equations are very similar to hose we had in the vector case, and we hence end up with a linear system of he form (18), with basically the same expressions:

$$A_{i,j} = (\psi_i, \psi_j), \tag{27}$$
$$b_i = (f, \psi_i)\,. \tag{28}$$

.2 The projection (or Galerkin) method

s in Section 1.2, the minimization of (e, e) is equivalent to

$$(e, v) = 0, \quad \forall v \in V\,. \tag{29}$$

his is known as a projection of a function f onto the subspace V. We may also all it a Galerkin method for approximating functions. Using the same reasoning s in (21)-(22), it follows that (29) is equivalent to

$$(e, \psi_i) = 0, \quad i \in \mathcal{I}_s\,. \tag{30}$$

nserting $e = f - u$ in this equation and ordering terms, as in the multi-imensional vector case, we end up with a linear system with a coefficient matrix 27) and right-hand side vector (28).

Whether we work with vectors in the plane, general vectors, or fu in function spaces, the least squares principle and the projection or G method are equivalent.

2.3 Example: linear approximation

Let us apply the theory in the previous section to a simple problem: parabola $f(x) = 10(x-1)^2 - 1$ for $x \in \Omega = [1, 2]$, find the best approx $u(x)$ in the space of all linear functions:

$$V = \mathrm{span}\,\{1, x\}\,.$$

With our notation, $\psi_0(x) = 1$, $\psi_1(x) = x$, and $N = 1$. We seek

$$u = c_0\psi_0(x) + c_1\psi_1(x) = c_0 + c_1 x,$$

where c_0 and c_1 are found by solving a 2×2 the linear system. The co matrix has elements

$$A_{0,0} = (\psi_0, \psi_0) = \int_1^2 1 \cdot 1\ \mathrm{d}x = 1,$$
$$A_{0,1} = (\psi_0, \psi_1) = \int_1^2 1 \cdot x\ \mathrm{d}x = 3/2,$$
$$A_{1,0} = A_{0,1} = 3/2,$$
$$A_{1,1} = (\psi_1, \psi_1) = \int_1^2 x \cdot x\ \mathrm{d}x = 7/3\,.$$

The corresponding right-hand side is

$$b_1 = (f, \psi_0) = \int_1^2 (10(x-1)^2 - 1) \cdot 1\ \mathrm{d}x = 7/3,$$
$$b_2 = (f, \psi_1) = \int_1^2 (10(x-1)^2 - 1) \cdot x\ \mathrm{d}x = 13/3\,.$$

Solving the linear system results in

$$c_0 = -38/3, \quad c_1 = 10,$$

and consequently

$$u(x) = 10x - \frac{38}{3}\,.$$

Figure 3 displays the parabola and its best approximation in the spa linear functions.

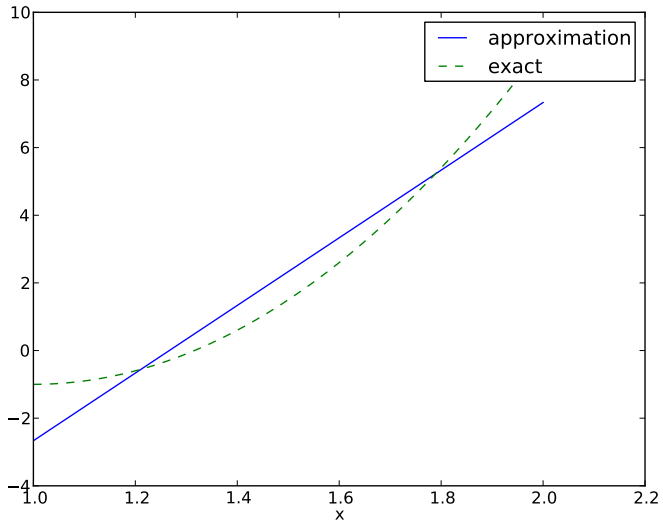


Figure 3: Best approximation of a parabola by a straight line.

.4 Implementation of the least squares method

ymbolic integration. The linear system can be computed either symboli-ally or numerically (a numerical integration rule is needed in the latter case). ere is a function for symbolic computation of the linear system, where $f(x)$ is iven as a `sympy` expression `f` involving the symbol `x`, `psi` is a list of expressions or $\{\psi_i\}_{i\in\mathcal{I}_s}$, and `Omega` is a 2-tuple/list holding the limits of the domain Ω:

```
import sympy as sp

def least_squares(f, psi, Omega):
    N = len(psi) - 1
    A = sp.zeros((N+1, N+1))
    b = sp.zeros((N+1, 1))
    x = sp.Symbol('x')
    for i in range(N+1):
        for j in range(i, N+1):
            A[i,j] = sp.integrate(psi[i]*psi[j],
                                  (x, Omega[0], Omega[1]))
            A[j,i] = A[i,j]
        b[i,0] = sp.integrate(psi[i]*f, (x, Omega[0], Omega[1]))
    c = A.LUsolve(b)
    u = 0
    for i in range(len(psi)):
        u += c[i,0]*psi[i]
    return u, c
```


Observe that we exploit the symmetry of the coefficient matrix: o upper triangular part is computed. Symbolic integration in `sympy` time consuming, and (roughly) halving the work has noticeable effect waiting time for the function to finish execution.

Fallback on numerical integration. Obviously, `sympy` mail fail to fully integrate $\int_\Omega \psi_i\psi_j dx$ and especially $\int_\Omega f\psi_i dx$ symbolically. There should extend the `least_squares` function such that it falls back on nu integration if the symbolic integration is unsuccessful. In the latter ca returned value from `sympy`'s `integrate` function is an object of type `In` We can test on this type and utilize the `mpmath` module in `sympy` to p numerical integration of high precision. Even when `sympy` manages to in symbolically, it can take an undesirable long time. We therefore inc argument `symbolic` that governs whether or not to try symbolic inte Here is the complete code of the improved version of function `least_s`

```
def least_squares(f, psi, Omega, symbolic=True):
    N = len(psi) - 1
    A = sp.zeros((N+1, N+1))
    b = sp.zeros((N+1, 1))
    x = sp.Symbol('x')
    for i in range(N+1):
        for j in range(i, N+1):
            integrand = psi[i]*psi[j]
            if symbolic:
                I = sp.integrate(integrand, (x, Omega[0], Omega[1
            if not symbolic or isinstance(I, sp.Integral):
                # Could not integrate symbolically,
                # fall back on numerical integration
                integrand = sp.lambdify([x], integrand)
                I = sp.mpmath.quad(integrand, [Omega[0], Omega[1]
            A[i,j] = A[j,i] = I

        integrand = psi[i]*f
        if symbolic:
            I = sp.integrate(integrand, (x, Omega[0], Omega[1]))
        if not symbolic or isinstance(I, sp.Integral):
            integrand = sp.lambdify([x], integrand)
            I = sp.mpmath.quad(integrand, [Omega[0], Omega[1]])
        b[i,0] = I
    c = A.LUsolve(b)  # symbolic solve
    # c is a sympy Matrix object, numbers are in c[i,0]
    u = sum(c[i,0]*psi[i] for i in range(len(psi)))
    return u, [c[i,0] for i in range(len(c))]
```

The function is found in the file `approx1D.py`.

Plotting the approximation. Comparing the given $f(x)$ and the imate $u(x)$ visually is done by the following function, which with th `sympy`'s `lambdify` tool converts a `sympy` expression to a Python func numerical computations:

```
lef comparison_plot(f, u, Omega, filename='tmp.pdf'):
    x = sp.Symbol('x')
    f = sp.lambdify([x], f, modules="numpy")
    u = sp.lambdify([x], u, modules="numpy")
    resolution = 401  # no of points in plot
    xcoor  = linspace(Omega[0], Omega[1], resolution)
    exact  = f(xcoor)
    approx = u(xcoor)
    plot(xcoor, approx)
    hold('on')
    plot(xcoor, exact)
    legend(['approximation', 'exact'])
    savefig(filename)
```

he `modules='numpy'` argument to `lambdify` is important if there are mathe-
atical functions, such as `sin` or `exp` in the symbolic expressions in `f` or `u`, and
ese mathematical functions are to be used with vector arguments, like `xcoor`
bove.

Both the `least_squares` and `comparison_plot` are found and coded in
e file `approx1D.py`[2]. The forthcoming examples on their use appear in
`x_approx1D.py`.

.5 Perfect approximation

et us use the code above to recompute the problem from Section 2.3 where we
ant to approximate a parabola. What happens if we add an element x^2 to the
asis and test what the best approximation is if V is the space of all parabolic
nctions? The answer is quickly found by running

```
>>> from approx1D import *
>>> x = sp.Symbol('x')
>>> f = 10*(x-1)**2-1
>>> u, c = least_squares(f=f, psi=[1, x, x**2], Omega=[1, 2])
>>> print u
10*x**2 - 20*x + 9
>>> print sp.expand(f)
10*x**2 - 20*x + 9
```

Now, what if we use $\psi_i(x) = x^i$ for $i = 0, 1, \ldots, N = 40$? The output from
east_squares gives $c_i = 0$ for $i > 2$, which means that the method finds the
erfect approximation.

In fact, we have a general result that if $f \in V$, the least squares and
rojection/Galerkin methods compute the exact solution $u = f$. The proof is
raightforward: if $f \in V$, f can be expanded in terms of the basis functions,
$= \sum_{j\in\mathcal{I}_s} d_j\psi_j$, for some coefficients $\{d_i\}_{i\in\mathcal{I}_s}$, and the right-hand side then has
ntries

$$b_i = (f, \psi_i) = \sum_{j\in\mathcal{I}_s} d_j(\psi_j, \psi_i) = \sum_{j\in\mathcal{I}_s} d_j A_{i,j}\,.$$

[2] `http://tinyurl.com/nm5587k/fem/approx1D.py`

The linear system $\sum_j A_{i,j}c_j = b_i$, $i \in \mathcal{I}_s$, is then

$$\sum_{j\in\mathcal{I}_s} c_j A_{i,j} = \sum_{j\in\mathcal{I}_s} d_j A_{i,j}, \quad i \in \mathcal{I}_s,$$

which implies that $c_i = d_i$ for $i \in \mathcal{I}_s$.

2.6 Ill-conditioning

The computational example in Section 2.5 applies the `least_squares` f
which invokes symbolic methods to calculate and solve the linear syste
correct solution $c_0 = 9, c_1 = -20, c_2 = 10, c_i = 0$ for $i \geq 3$ is perfectly re

Suppose we convert the matrix and right-hand side to floating-poin
and then solve the system using finite-precision arithmetics, which is w
will (almost) always do in real life. This time we get astonishing resu
to about $N = 7$ we get a solution that is reasonably close to the ex
Increasing N shows that seriously wrong coefficients are computed. I
a table showing the solution of the linear system arising from approx
a parabola by functions on the form $u(x) = c_0 + c_1x + c_2x^2 + \cdots +$
Analytically, we know that $c_j = 0$ for $j > 2$, but numerically we may ge
for $j > 2$.

exact	`sympy`	`numpy32`	`numpy64`
9	9.62	5.57	8.98
-20	-23.39	-7.65	-19.93
10	17.74	-4.50	9.96
0	-9.19	4.13	-0.26
0	5.25	2.99	0.72
0	0.18	-1.21	-0.93
0	-2.48	-0.41	0.73
0	1.81	-0.013	-0.36
0	-0.66	0.08	0.11
0	0.12	0.04	-0.02
0	-0.001	-0.02	0.002

The exact value of c_j, $j = 0, 1, \ldots, 10$, appears in the first column w
other columns correspond to results obtained by three different method

- Column 2: The matrix and vector are converted to the data st
  `sympy.mpmath.fp.matrix` and the `sympy.mpmath.fp.lu_solve` f
  is used to solve the system.
- Column 3: The matrix and vector are converted to `numpy` arrays w
  type `numpy.float32` (single precision floating-point number) anc
  by the `numpy.linalg.solve` function.
- Column 4: As column 3, but the data type is `numpy.float64`
  precision floating-point number).

/e see from the numbers in the table that double precision performs much better
ıan single precision. Nevertheless, when plotting all these solutions the curves
annot be visually distinguished (!). This means that the approximations look
erfect, despite the partially very wrong values of the coefficients.

Increasing N to 12 makes the numerical solver in `numpy` abort with the
ıessage: "matrix is numerically singular". A matrix has to be non-singular to be
ıvertible, which is a requirement when solving a linear system. Already when
ıe matrix is close to singular, it is *ill-conditioned*, which here implies that the
ımerical solution algorithms are sensitive to round-off errors and may produce
ery) inaccurate results.

The reason why the coefficient matrix is nearly singular and ill-conditioned
that our basis functions $\psi_i(x) = x^i$ are nearly linearly dependent for large i.
hat is, x^i and x^{i+1} are very close for i not very small. This phenomenon is
lustrated in Figure 4. There are 15 lines in this figure, but only half of them
re visually distinguishable. Almost linearly dependent basis functions give rise
› an ill-conditioned and almost singular matrix. This fact can be illustrated by
›mputing the determinant, which is indeed very close to zero (recall that a zero
eterminant implies a singular and non-invertible matrix): 10^{-65} for $N = 10$ and
)$^{-92}$ for $N = 12$. Already for $N = 28$ the numerical determinant computation
eturns a plain zero.

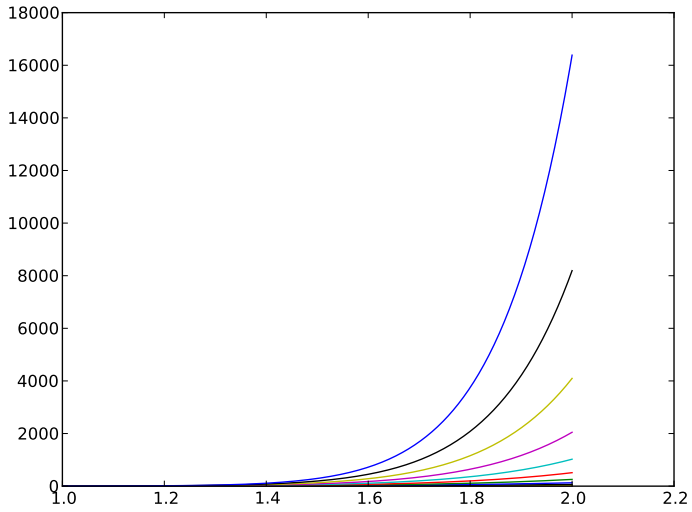


Figure 4: The 15 first basis functions x^i, $i = 0, \ldots, 14$.

On the other hand, the double precision `numpy` solver do run for $N = 100$,
esulting in answers that are not significantly worse than those in the table

above, and large powers are associated with small coefficients (e.g., c_j
for $10 \leq j \leq 20$ and $c < 10^{-5}$ for $j > 20$). Even for $N = 100$ the approx
still lies on top of the exact curve in a plot (!).

The conclusion is that visual inspection of the quality of the approx
may not uncover fundamental numerical problems with the computatior
ever, numerical analysts have studied approximations and ill-conditio
decades, and it is well known that the basis $\{1, x, x^2, x^3, \ldots, \}$ is a ba
The best basis from a matrix conditioning point of view is to have ort
functions such that $(\psi_i, \psi_j) = 0$ for $i \neq j$. There are many known sets of
onal polynomials and other functions. The functions used in the finite
methods are almost orthogonal, and this property helps to avoid proble
solving matrix systems. Almost orthogonal is helpful, but not enoug
it comes to partial differential equations, and ill-conditioning of the co
matrix is a theme when solving large-scale matrix systems arising fro
element discretizations.

2.7 Fourier series

A set of sine functions is widely used for approximating functions (the s
also orthogonal as explained more in Section 2.6). Let us take

$$V = \text{span}\,\{\sin \pi x, \sin 2\pi x, \ldots, \sin(N+1)\pi x\}\,.$$

That is,

$$\psi_i(x) = \sin((i+1)\pi x), \quad i \in \mathcal{I}_s\,.$$

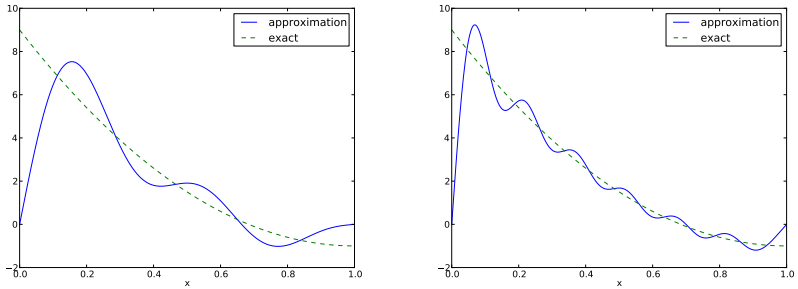
An approximation to the $f(x)$ function from Section 2.3 can then be co
by the `least_squares` function from Section 2.4:

```
N = 3
from sympy import sin, pi
x = sp.Symbol('x')
psi = [sin(pi*(i+1)*x) for i in range(N+1)]
f = 10*(x-1)**2 - 1
Omega = [0, 1]
u, c = least_squares(f, psi, Omega)
comparison_plot(f, u, Omega)
```

Figure 5 (left) shows the oscillatory approximation of $\sum_{j=0}^{N} c_j \sin((j$
when $N = 3$. Changing N to 11 improves the approximation considera
Figure 5 (right).

There is an error $f(0) - u(0) = 9$ at $x = 0$ in Figure 5 regardless of h
N is, because all $\psi_i(0) = 0$ and hence $u(0) = 0$. We may help the approx
to be correct at $x = 0$ by seeking

$$u(x) = f(0) + \sum_{j \in \mathcal{I}_s} c_j \psi_j(x)\,.$$



igure 5: Best approximation of a parabola by a sum of 3 (left) and 11 (right) ne functions.

owever, this adjustment introduces a new problem at $x = 1$ since we now get n error $f(1) - u(1) = f(1) - 0 = -1$ at this point. A more clever adjustment is o replace the $f(0)$ term by a term that is $f(0)$ at $x = 0$ and $f(1)$ at $x = 1$. A mple linear combination $f(0)(1 - x) + xf(1)$ does the job:

$$u(x) = f(0)(1 - x) + xf(1) + \sum_{j \in \mathcal{I}_s} c_j \psi_j(x)\,. \tag{40}$$

his adjustment of u alters the linear system slightly. In the general case, we set

$$u(x) = B(x) + \sum_{j \in \mathcal{I}_s} c_j \psi_j(x),$$

nd the linear system becomes

$$\sum_{j \in \mathcal{I}_s} (\psi_i, \psi_j) c_j = (f - B, \psi_i), \quad i \in \mathcal{I}_s\,.$$

he calculations can still utilize the `least_squares` or `least_squares_orth` nctions, but solve for $u - b$:

```
f0 = 0;  f1 = -1
B = f0*(1-x) + x*f1
u_sum, c = least_squares_orth(f-b, psi, Omega)
u = B + u_sum
```

Figure 6 shows the result of the technique for ensuring right boundary values. ven 3 sines can now adjust the $f(0)(1-x)+xf(1)$ term such that u approximates ne parabola really well, at least visually.

.8 Orthogonal basis functions

he choice of sine functions $\psi_i(x) = \sin((i + 1)\pi x)$ has a great computational dvantage: on $\Omega = [0, 1]$ these basis functions are *orthogonal*, implying that $_{i,j} = 0$ if $i \neq j$. This result is realized by trying

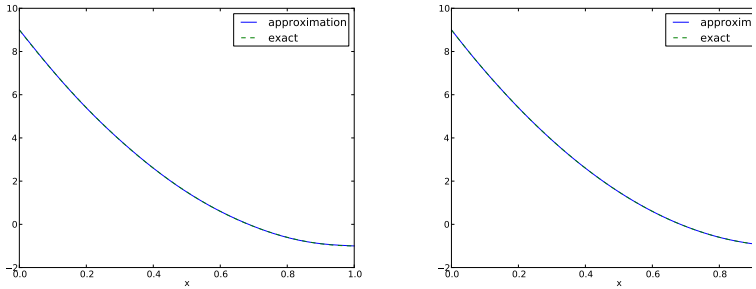


Figure 6: Best approximation of a parabola by a sum of 3 (left) and 1 sine functions with a boundary term.

```
integrate(sin(j*pi*x)*sin(k*pi*x), x, 0, 1)
```

in WolframAlpha[3] (avoid `i` in the integrand as this symbol means the im unit $\sqrt{-1}$). Also by asking WolframAlpha about $\int_0^1 \sin^2(j\pi x)\,\mathrm{d}x$, we fi equal 1/2. With a diagonal matrix we can easily solve for the coeffic hand:

$$c_i = 2\int_0^1 f(x)\sin((i + 1)\pi x)\,\mathrm{d}x, \quad i \in \mathcal{I}_s,$$

which is nothing but the classical formula for the coefficients of the Fou series of $f(x)$ on $[0, 1]$. In fact, when V contains the basic functions u Fourier series expansion, the approximation method derived in Section 2 in the classical Fourier series for $f(x)$ (see Exercise 8 for details).

With orthogonal basis functions we can make the `least_squares` f (much) more efficient since we know that the matrix is diagonal and c diagonal elements need to be computed:

```
def least_squares_orth(f, psi, Omega):
    N = len(psi) - 1
    A = [0]*(N+1)
    b = [0]*(N+1)
    x = sp.Symbol('x')
    for i in range(N+1):
        A[i] = sp.integrate(psi[i]**2, (x, Omega[0], Omega[1]))
        b[i] = sp.integrate(psi[i]*f,  (x, Omega[0], Omega[1]))
    c = [b[i]/A[i] for i in range(len(b))]
    u = 0
    for i in range(len(psi)):
        u += c[i]*psi[i]
    return u, c
```

As mentioned in Section 2.4, symbolic integration may fail or take v time. It is therefore natural to extend the implementation above with a

[3] `http://wolframalpha.com`

here we can choose between symbolic and numerical integration and fall back n the latter if the former fails:

```
def least_squares_orth(f, psi, Omega, symbolic=True):
    N = len(psi) - 1
    A = [0]*(N+1)       # plain list to hold symbolic expressions
    b = [0]*(N+1)
    x = sp.Symbol('x')
    for i in range(N+1):
        # Diagonal matrix term
        A[i] = sp.integrate(psi[i]**2, (x, Omega[0], Omega[1]))

        # Right-hand side term
        integrand = psi[i]*f
        if symbolic:
            I = sp.integrate(integrand,  (x, Omega[0], Omega[1]))
        if not symbolic or isinstance(I, sp.Integral):
            print 'numerical integration of', integrand
            integrand = sp.lambdify([x], integrand)
            I = sp.mpmath.quad(integrand, [Omega[0], Omega[1]])
        b[i] = I
    c = [b[i]/A[i] for i in range(len(b))]
    u = 0
    u = sum(c[i,0]*psi[i] for i in range(len(psi)))
    return u, c
```

his function is found in the file `approx1D.py`. Observe that we here assume at $\int_\Omega \varphi_i^2 dx$ can always be symbolically computed, which is not an unreasonable ssumption when the basis functions are orthogonal, but there is no guarantee, so n improved version of the function above would implement numerical integration lso for the `A[i,i]` term.

.9 Numerical computations

ometimes the basis functions ψ_i and/or the function f have a nature that akes symbolic integration CPU-time consuming or impossible. Even though we nplemented a fallback on numerical integration of $\int f\varphi_i dx$ considerable time ight be required by `sympy` in the attempt to integrate symbolically. Therefore, will be handy to have function for fast *numerical integration and numerical olution of the linear system*. Below is such a method. It requires Python nctions `f(x)` and `psi(x,i)` for $f(x)$ and $\psi_i(x)$ as input. The output is a mesh nction with values `u` on the mesh with points in the array `x`. Three numerical tegration methods are offered: `scipy.integrate.quad` (precision set to 10^{-8}), `ympy.mpmath.quad` (high precision), and a Trapezoidal rule based on the points `x`.

```
def least_squares_numerical(f, psi, N, x,
                            integration_method='scipy',
                            orthogonal_basis=False):
    import scipy.integrate
    A = np.zeros((N+1, N+1))
    b = np.zeros(N+1)
    Omega = [x[0], x[-1]]
```



```
    dx = x[1] - x[0]

    for i in range(N+1):
        j_limit = i+1 if orthogonal_basis else N+1
        for j in range(i, j_limit):
            print '(%d,%d)' % (i, j)
            if integration_method == 'scipy':
                A_ij = scipy.integrate.quad(
                    lambda x: psi(x,i)*psi(x,j),
                    Omega[0], Omega[1], epsabs=1E-9, epsrel=1E-9)
            elif integration_method == 'sympy':
                A_ij = sp.mpmath.quad(
                    lambda x: psi(x,i)*psi(x,j),
                    [Omega[0], Omega[1]])
            else:
                values = psi(x,i)*psi(x,j)
                A_ij = trapezoidal(values, dx)
            A[i,j] = A[j,i] = A_ij

        if integration_method == 'scipy':
            b_i = scipy.integrate.quad(
                lambda x: f(x)*psi(x,i), Omega[0], Omega[1],
                epsabs=1E-9, epsrel=1E-9)[0]
        elif integration_method == 'sympy':
            b_i = sp.mpmath.quad(
                lambda x: f(x)*psi(x,i), [Omega[0], Omega[1]])
        else:
            values = f(x)*psi(x,i)
            b_i = trapezoidal(values, dx)
        b[i] = b_i

    c = b/np.diag(A) if orthogonal_basis else np.linalg.solve(A,
    u = sum(c[i]*psi(x, i) for i in range(N+1))
    return u, c

def trapezoidal(values, dx):
    """Integrate values by the Trapezoidal rule (mesh size dx)."""
    return dx*(np.sum(values) - 0.5*values[0] - 0.5*values[-1])
```

Here is an example on calling the function:

```
from numpy import linspace, tanh, pi

def psi(x, i):
    return sin((i+1)*x)

x = linspace(0, 2*pi, 501)
N = 20
u, c = least_squares_numerical(lambda x: tanh(x-pi), psi, N, x,
                               orthogonal_basis=True)
```

2.10 The interpolation (or collocation) method

The principle of minimizing the distance between u and f is an intuit of computing a best approximation $u \in V$ to f. However, there ar approaches as well. One is to demand that $u(x_i) = f(x_i)$ at some selecte x_i, $i \in \mathcal{I}_s$:

$$u(x_i) = \sum_{j\in\mathcal{I}_s} c_j\psi_j(x_i) = f(x_i), \quad i \in \mathcal{I}_s\,. \tag{42}$$

his criterion also gives a linear system with $N+1$ unknown coefficients $\{c_i\}_{i\in\mathcal{I}_s}$:

$$\sum_{j\in\mathcal{I}_s} A_{i,j}c_j = b_i, \quad i \in \mathcal{I}_s, \tag{43}$$

ith

$$A_{i,j} = \psi_j(x_i), \tag{44}$$
$$b_i = f(x_i)\,. \tag{45}$$

his time the coefficient matrix is not symmetric because $\psi_j(x_i) \neq \psi_i(x_j)$ in eneral. The method is often referred to as an *interpolation method* since some oint values of f are given ($f(x_i)$) and we fit a continuous function u that goes rough the $f(x_i)$ points. In this case the x_i points are called *interpolation oints*. When the same approach is used to approximate differential equations, ne usually applies the name *collocation method* and x_i are known as *collocation oints*.

Given f as a `sympy` symbolic expression `f`, $\{\psi_i\}_{i\in\mathcal{I}_s}$ as a list `psi`, and a set f points $\{x_i\}_{i\in\mathcal{I}_s}$ as a list or array `points`, the following Python function sets p and solves the matrix system for the coefficients $\{c_i\}_{i\in\mathcal{I}_s}$:

```
lef interpolation(f, psi, points):
    N = len(psi) - 1
    A = sp.zeros((N+1, N+1))
    b = sp.zeros((N+1, 1))
    x = sp.Symbol('x')
    # Turn psi and f into Python functions
    psi = [sp.lambdify([x], psi[i]) for i in range(N+1)]
    f = sp.lambdify([x], f)
    for i in range(N+1):
        for j in range(N+1):
            A[i,j] = psi[j](points[i])
        b[i,0] = f(points[i])
    c = A.LUsolve(b)
    u = 0
    for i in range(len(psi)):
        u += c[i,0]*psi[i](x)
    return u
```

he `interpolation` function is a part of the `approx1D` module.

We found it convenient in the above function to turn the expressions `f` nd `psi` into ordinary Python functions of `x`, which can be called with `float` alues in the list `points` when building the matrix and the right-hand side. he alternative is to use the `subs` method to substitute the `x` variable in an xpression by an element from the `points` list. The following session illustrates oth approaches in a simple setting:


```
>>> from sympy import *
>>> x = Symbol('x')
>>> e = x**2              # symbolic expression involving x
>>> p = 0.5               # a value of x
>>> v = e.subs(x, p)      # evaluate e for x=p
>>> v
0.250000000000000
>>> type(v)
sympy.core.numbers.Float
>>> e = lambdify([x], e)  # make Python function of e
>>> type(e)
>>> function
>>> v = e(p)              # evaluate e(x) for x=p
>>> v
0.25
>>> type(v)
float
```

A nice feature of the interpolation or collocation method is that it computing integrals. However, one has to decide on the location of the x A simple, yet common choice, is to distribute them uniformly throughc

Example. Let us illustrate the interpolation or collocation method by imating our parabola $f(x) = 10(x-1)^2 - 1$ by a linear function on Ω using two collocation points $x_0 = 1 + 1/3$ and $x_1 = 1 + 2/3$:

```
f = 10*(x-1)**2 - 1
psi = [1, x]
Omega = [1, 2]
points = [1 + sp.Rational(1,3), 1 + sp.Rational(2,3)]
u = interpolation(f, psi, points)
comparison_plot(f, u, Omega)
```

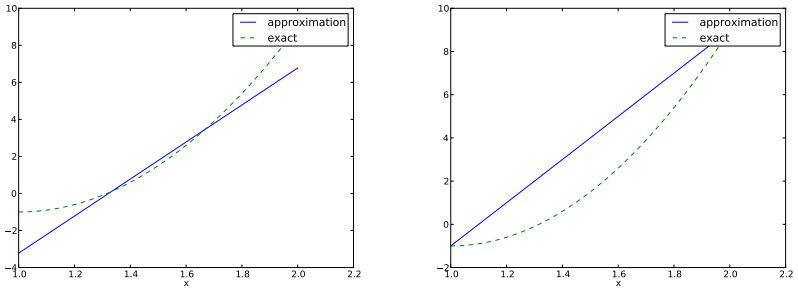
The resulting linear system becomes

$$\begin{pmatrix} 1 & 4/3 \\ 1 & 5/3 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 1/9 \\ 31/9 \end{pmatrix}$$

with solution $c_0 = -119/9$ and $c_1 = 10$. Figure 7 (left) shows the r approximation $u = -119/9 + 10x$. We can easily test other interpolatio say $x_0 = 1$ and $x_1 = 2$. This changes the line quite significantly, see F (right).

2.11 Lagrange polynomials

In Section 2.7 we explain the advantage with having a diagonal matrix: f for the coefficients $\{c_i\}_{i\in\mathcal{I}_s}$ can then be derived by hand. For an interpc collocation method a diagonal matrix implies that $\psi_j(x_i) = 0$ if $i \neq$ set of basis functions $\psi_i(x)$ with this property is the *Lagrange inter polynomials*, or just *Lagrange polynomials*. (Although the functions are after Lagrange, they were first discovered by Waring in 1779, rediscov



igure 7: Approximation of a parabola by linear functions computed by two
ıterpolation points: 4/3 and 5/3 (left) versus 1 and 2 (right).

uler in 1783, and published by Lagrange in 1795.) The Lagrange polynomials
ave the form

$$\psi_i(x) = \prod_{j=0, j\neq i}^{N} \frac{x - x_j}{x_i - x_j} = \frac{x - x_0}{x_i - x_0} \cdots \frac{x - x_{i-1}}{x_i - x_{i-1}} \frac{x - x_{i+1}}{x_i - x_{i+1}} \cdots \frac{x - x_N}{x_i - x_N}, \quad (46)$$

ır $i \in \mathcal{I}_s$. We see from (46) that all the ψ_i functions are polynomials of degree
/ which have the property

$$\psi_i(x_s) = \delta_{is}, \quad \delta_{is} = \begin{cases} 1, & i = s, \\ 0, & i \neq s, \end{cases} \quad (47)$$

hen x_s is an interpolation/collocation point. Here we have used the *Kronecker*
elta symbol δ_{is}. This property implies that $A_{i,j} = 0$ for $i \neq j$ and $A_{i,j} = 1$
hen $i = j$. The solution of the linear system is them simply

$$c_i = f(x_i), \quad i \in \mathcal{I}_s, \quad (48)$$

nd

$$u(x) = \sum_{j \in \mathcal{I}_s} f(x_i)\psi_i(x)\,. \quad (49)$$

The following function computes the Lagrange interpolating polynomial $\psi_i(x)$,
iven the interpolation points $x_0, \ldots, x_N$ in the list or array `points`:

```
lef Lagrange_polynomial(x, i, points):
    p = 1
    for k in range(len(points)):
        if k != i:
            p *= (x - points[k])/(points[i] - points[k])
    return p
```

he next function computes a complete basis using equidistant points throughout
:


```
def Lagrange_polynomials_01(x, N):
    if isinstance(x, sp.Symbol):
        h = sp.Rational(1, N-1)
    else:
        h = 1.0/(N-1)
    points = [i*h for i in range(N)]
    psi = [Lagrange_polynomial(x, i, points) for i in range(N)]
    return psi, points
```

When `x` is an `sp.Symbol` object, we let the spacing between the interp
points, `h`, be a `sympy` rational number for nice end results in the f
for ψ_i. The other case, when `x` is a plain Python `float`, signifies nu
computing, and then we let `h` be a floating-point number. Observe t
`Lagrange_polynomial` function works equally well in the symbolic and nu
case - just think of `x` being an `sp.Symbol` object or a Python `float`.
interactive session illustrates the difference between symbolic and nu
computing of the basis functions and points:

```
>>> import sympy as sp
>>> x = sp.Symbol('x')
>>> psi, points = Lagrange_polynomials_01(x, N=3)
>>> points
[0, 1/2, 1]
>>> psi
[(1 - x)*(1 - 2*x), 2*x*(2 - 2*x), -x*(1 - 2*x)]

>>> x = 0.5  # numerical computing
>>> psi, points = Lagrange_polynomials_01(x, N=3)
>>> points
[0.0, 0.5, 1.0]
>>> psi
[-0.0, 1.0, 0.0]
```

The Lagrange polynomials are very much used in finite element methods
of their property (47).

Approximation of a polynomial. The Galerkin or least squares metl
to an exact approximation if f lies in the space spanned by the basis funct
could be interest to see how the interpolation method with Lagrange poly
as basis is able to approximate a polynomial, e.g., a parabola. Running

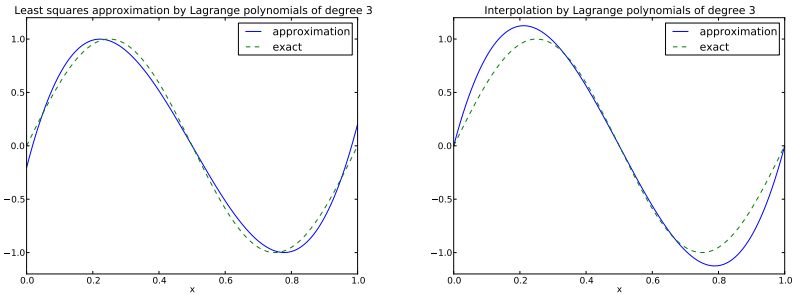
```
for N in 2, 4, 5, 6, 8, 10, 12:
    f = x**2
    psi, points = Lagrange_polynomials_01(x, N)
    u = interpolation(f, psi, points)
```

shows the result that up to `N=4` we achieve an exact approximation, a
round-off errors start to grow, such that `N=15` leads to a 15-degree pol
for u where the coefficients in front of x^r for $r > 2$ are of size 10^{-5} and

Successful example. Trying out the Lagrange polynomial basis for
mating $f(x) = \sin 2\pi x$ on $\Omega = [0, 1]$ with the least squares and the interp
techniques can be done by

```
x = sp.Symbol('x')
f = sp.sin(2*sp.pi*x)
psi, points = Lagrange_polynomials_01(x, N)
Omega=[0, 1]
u, c = least_squares(f, psi, Omega)
comparison_plot(f, u, Omega)
u, c = interpolation(f, psi, points)
comparison_plot(f, u, Omega)
```

igure 8 shows the results. There is little difference between the least squares and ne interpolation technique. Increasing N gives visually better approximations.



igure 8: Approximation via least squares (left) and interpolation (right) of a ne function by Lagrange interpolating polynomials of degree 3.

ess successful example. The next example concerns interpolating $f(x) = |1 - 2x|$ on $\Omega = [0, 1]$ using Lagrange polynomials. Figure 9 shows a peculiar ffect: the approximation starts to oscillate more and more as N grows. This umerical artifact is not surprising when looking at the individual Lagrange olynomials. Figure 10 shows two such polynomials, $\psi_2(x)$ and $\psi_7(x)$, both of egree 11 and computed from uniformly spaced points $x_{x_i} = i/11$, $i = 0, \ldots, 11$, narked with circles. We clearly see the property of Lagrange polynomials: $_2(x_i) = 0$ and $\psi_7(x_i) = 0$ for all i, except $\psi_2(x_2) = 1$ and $\psi_7(x_7) = 1$. The ost striking feature, however, is the significant oscillation near the boundary. he reason is easy to understand: since we force the functions to zero at so many oints, a polynomial of high degree is forced to oscillate between the points. The henomenon is named *Runge's phenomenon* and you can read a more detailed xplanation on Wikipedia[4].

emedy for strong oscillations. The oscillations can be reduced by a more ever choice of interpolation points, called the *Chebyshev nodes*:

$$x_i = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{2i+1}{2(N+1)}pi\right), \quad i = 0\ldots, N, \tag{50}$$

[4] http://en.wikipedia.org/wiki/Runge%27s_phenomenon

on the interval $\Omega = [a, b]$. Here is a flexible version of the Lagrange_polyn function above, valid for any interval $\Omega = [a, b]$ and with the possibility t ate both uniformly distributed points and Chebyshev nodes:

```
def Lagrange_polynomials(x, N, Omega, point_distribution='uniform
    if point_distribution == 'uniform':
        if isinstance(x, sp.Symbol):
            h = sp.Rational(Omega[1] - Omega[0], N)
        else:
            h = (Omega[1] - Omega[0])/float(N)
        points = [Omega[0] + i*h for i in range(N+1)]
    elif point_distribution == 'Chebyshev':
        points = Chebyshev_nodes(Omega[0], Omega[1], N)
    psi = [Lagrange_polynomial(x, i, points) for i in range(N+1)]
    return psi, points

def Chebyshev_nodes(a, b, N):
    from math import cos, pi
    return [0.5*(a+b) + 0.5*(b-a)*cos(float(2*i+1)/(2*N+1))*pi) \
            for i in range(N+1)]
```

All the functions computing Lagrange polynomials listed above are foun module file Lagrange.py. Figure 11 shows the improvement of using Ch nodes (compared with Figure 9). The reason is that the corresponding L polynomials have much smaller oscillations as seen in Figure 12 (compa Figure 10).

Another cure for undesired oscillation of higher-degree interpolati nomials is to use lower-degree Lagrange polynomials on many small pa the domain, which is the idea pursued in the finite element method. stance, linear Lagrange polynomials on $[0, 1/2]$ and $[1/2, 1]$ would yield a approximation to $f(x) = |1 - 2x|$ on $\Omega = [0, 1]$ since f is piecewise linea

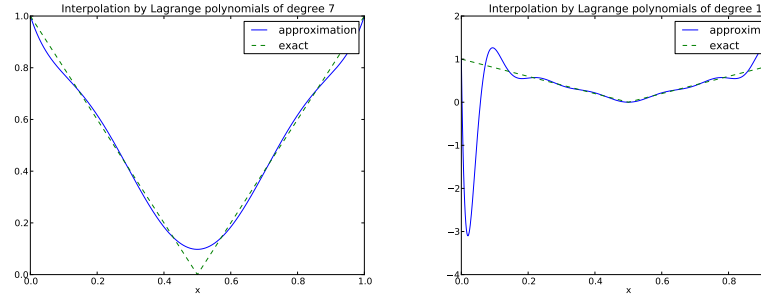
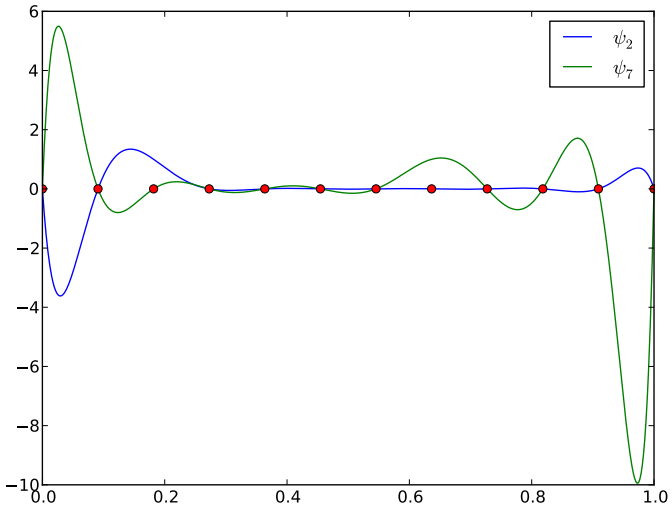
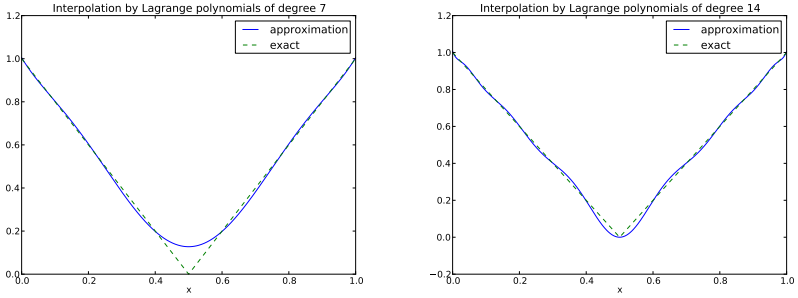


Figure 9: Interpolation of an absolute value function by Lagrange poly and uniformly distributed interpolation points: degree 7 (left) and 14 (

How does the least squares or projection methods work with L polynomials? We can just call the least_squares function, but sy problems integrating the $f(x) = |1 - 2x|$ function times a polynomia need to fallback on numerical integration.



igure 10: Illustration of the oscillatory behavior of two Lagrange polynomials ased on 12 uniformly spaced points (marked by circles).



igure 11: Interpolation of an absolute value function by Lagrange polynomials nd Chebyshev nodes as interpolation points: degree 7 (left) and 14 (right).

```
lef least_squares(f, psi, Omega):
    N = len(psi) - 1
    A = sp.zeros((N+1, N+1))
    b = sp.zeros((N+1, 1))
    x = sp.Symbol('x')
    for i in range(N+1):
        for j in range(i, N+1):
            integrand = psi[i]*psi[j]
            I = sp.integrate(integrand, (x, Omega[0], Omega[1]))
            if isinstance(I, sp.Integral):
```

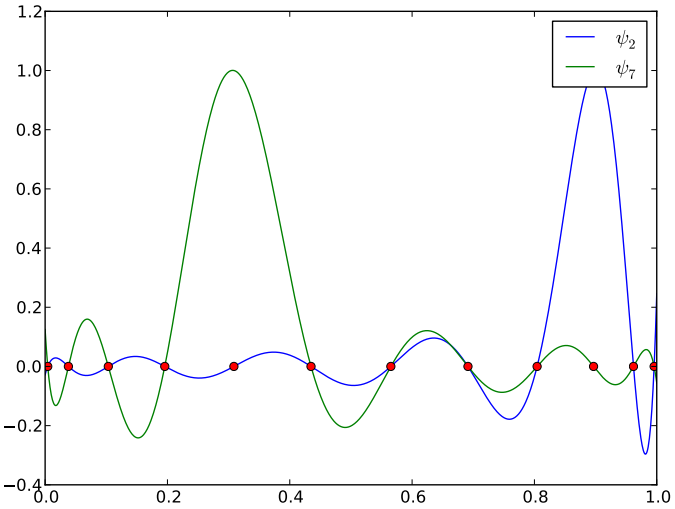



Figure 12: Illustration of the less oscillatory behavior of two Lagrange mials based on 12 Chebyshev points (marked by circles).

```
                # Could not integrate symbolically, fallback
                # on numerical integration with mpmath.quad
                integrand = sp.lambdify([x], integrand)
                I = sp.mpmath.quad(integrand, [Omega[0], Omega[1]
            A[i,j] = A[j,i] = I
        integrand = psi[i]*f
        I = sp.integrate(integrand, (x, Omega[0], Omega[1]))
        if isinstance(I, sp.Integral):
            integrand = sp.lambdify([x], integrand)
            I = sp.mpmath.quad(integrand, [Omega[0], Omega[1]])
        b[i,0] = I
    c = A.LUsolve(b)
    u = 0
    for i in range(len(psi)):
        u += c[i,0]*psi[i]
    return u
```

3 Finite element basis functions

The specific basis functions exemplified in Section 2 are in general nonzer entire domain Ω, see Figure 13 for an example where we plot $\psi_0(x) =$ and $\psi_1(x) = \sin 2\pi x$ together with a possible sum $u(x) = 4\psi_0(x) -$ We shall now turn the attention to basis functions that have *compact* meaning that they are nonzero on only a small portion of Ω. Morec

ıall restrict the functions to be *piecewise polynomials*. This means that the ɔmain is split into subdomains and the function is a polynomial on one or more ıbdomains, see Figure 14 for a sketch involving locally defined hat functions ıat make $u = \sum_j c_j \psi_j$ piecewise linear. At the boundaries between subdomains ıe normally forces continuity of the function only so that when connecting two olynomials from two subdomains, the derivative becomes discontinuous. These ɣpe of basis functions are fundamental in the finite element method.

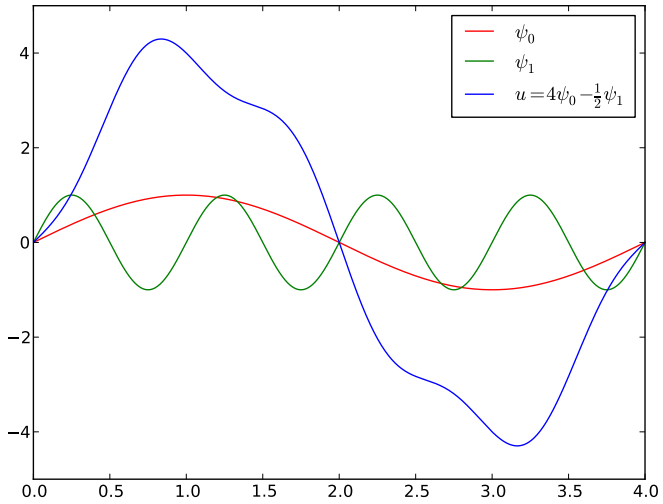


Figure 13: A function resulting from adding two sine basis functions.

We first introduce the concepts of elements and nodes in a simplistic fashion s often met in the literature. Later, we shall generalize the concept of an ement, which is a necessary step to treat a wider class of approximations within ıe family of finite element methods. The generalization is also compatible with ıe concepts used in the FEniCS[5] finite element software.

.1 Elements and nodes

et us divide the interval Ω on which f and u are defined into non-overlapping ıbintervals $\Omega^{(e)}$, $e = 0, \ldots, N_e$:

$$\Omega = \Omega^{(0)} \cup \cdots \cup \Omega^{(N_e)}. \tag{51}$$

/e shall for now refer to $\Omega^{(e)}$ as an *element*, having number e. On each element e introduce a set of points called *nodes*. For now we assume that the nodes are

[5] `http://fenicsproject.org`

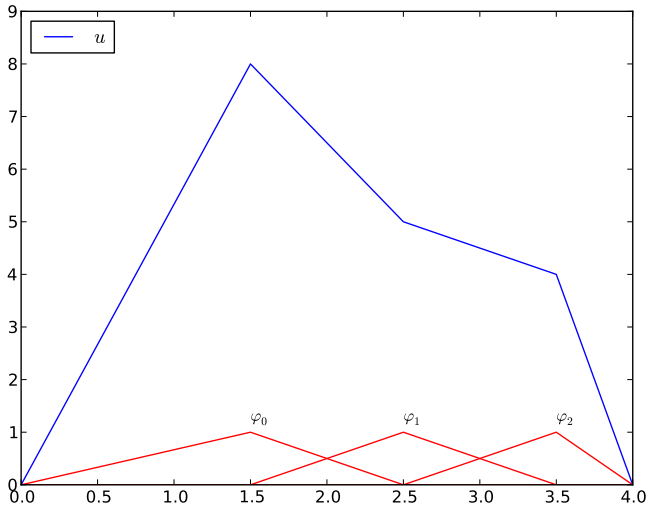


Figure 14: A function resulting from adding three local piecewise line functions.

uniformly spaced throughout the element and that the boundary point elements are also nodes. The nodes are given numbers both within an and in the global domain. These are referred to as *local* and *global* node r respectively. Figure 15 shows element boundaries with small vertical line as small disks, element numbers in circles, and global node numbers ur nodes.

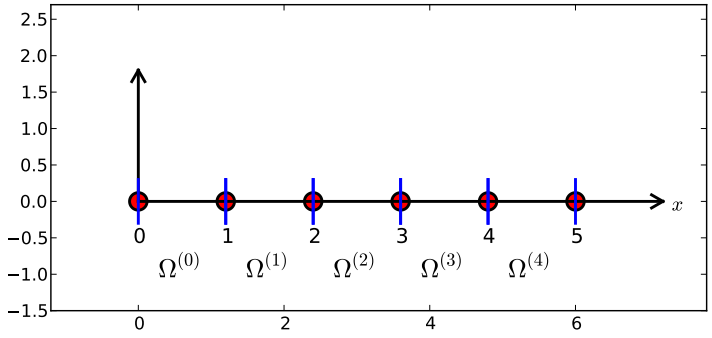


Figure 15: Finite element mesh with 5 elements and 6 nodes.

Nodes and elements uniquely define a *finite element mesh*, which is our iscrete representation of the domain in the computations. A common special ase is that of a *uniformly partitioned mesh* where each element has the same ngth and the distance between nodes is constant.

xample. On $\Omega = [0, 1]$ we may introduce two elements, $\Omega^{(0)} = [0, 0.4]$ and $^{(1)} = [0.4, 1]$. Furthermore, let us introduce three nodes per element, equally paced within each element. Figure 16 shows the mesh. The three nodes in ement number 0 are $x_0 = 0$, $x_1 = 0.2$, and $x_2 = 0.4$. The local and global node umbers are here equal. In element number 1, we have the local nodes $x_0 = 0.4$, $_1 = 0.7$, and $x_2 = 1$ and the corresponding global nodes $x_2 = 0.4$, $x_3 = 0.7$, nd $x_4 = 1$. Note that the global node $x_2 = 0.4$ is shared by the two elements.

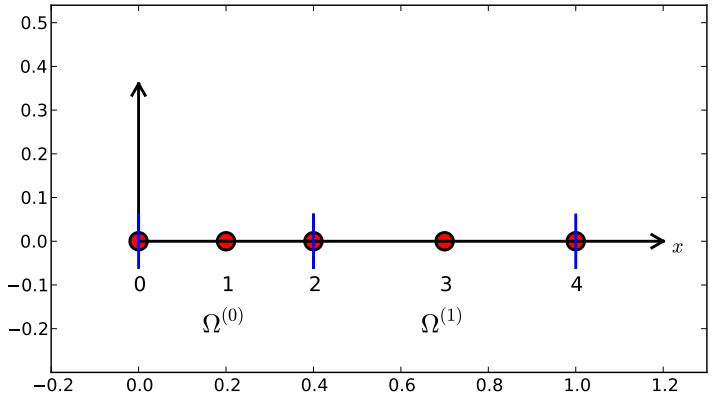


Figure 16: Finite element mesh with 2 elements and 5 nodes.

For the purpose of implementation, we introduce two lists or arrays: `nodes` r storing the coordinates of the nodes, with the global node numbers as indices, nd `elements` for holding the global node numbers in each element, with the ocal node numbers as indices. The `nodes` and `elements` lists for the sample esh above take the form

```
odes = [0, 0.2, 0.4, 0.7, 1]
lements = [[0, 1, 2], [2, 3, 4]]
```

ooking up the coordinate of local node number 2 in element 1 is here done by `odes[elements[1][2]]` (recall that nodes and elements start their numbering t 0).

The numbering of elements and nodes does not need to be regular. Figure 17 ows and example corresponding to

```
odes = [1.5, 5.5, 4.2, 0.3, 2.2, 3.1]
lements = [[2, 1], [4, 5], [0, 4], [3, 0], [5, 2]]
```

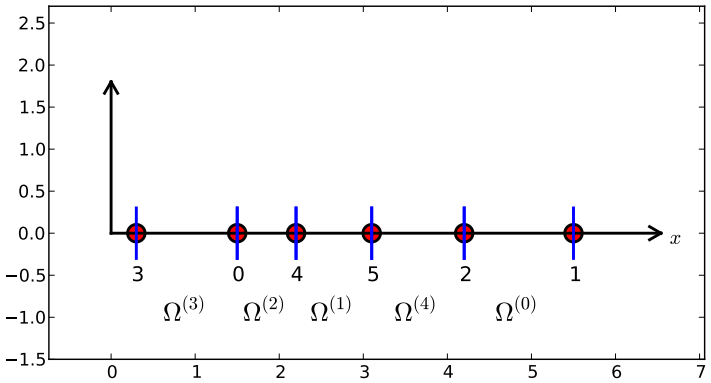



Figure 17: Example on irregular numbering of elements and node

3.2 The basis functions

Construction principles. Finite element basis functions are in this ognized by the notation $\varphi_i(x)$, where the index now in the beginning corr to a global node number. In the current approximation problem we shal take $\psi_i = \varphi_i$.

Let i be the global node number corresponding to local node r in number e. The finite element basis functions φ_i are now defined as foll

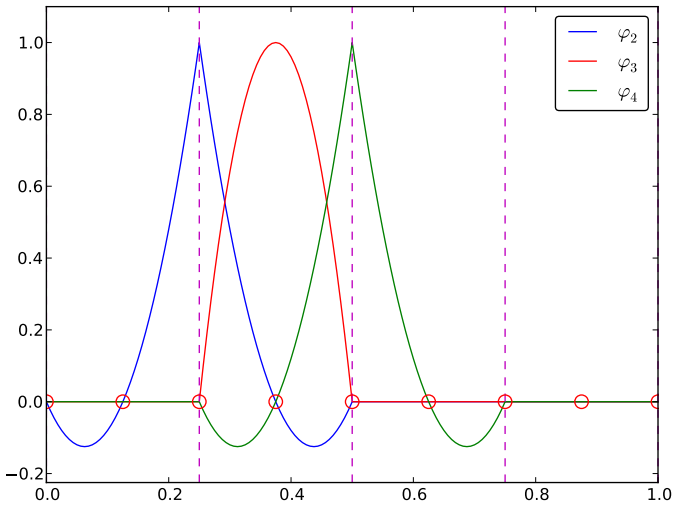
- If local node number r is not on the boundary of the element, tal to be the Lagrange polynomial that is 1 at the local node numb zero at all other nodes in the element. On all other elements, φ_i =
- If local node number r is on the boundary of the element, let φ_i b up of the Lagrange polynomial over element e that is 1 at node i, c with the Lagrange polynomial over element $e + 1$ that is also 1 at On all other elements, $\varphi_i = 0$.

A visual impression of three such basis functions are given in Figure 18

Properties of φ_i. The construction of basis functions according to th ples above lead to two important properties of $\varphi_i(x)$. First,

$$\varphi_i(x_j) = \delta_{ij}, \quad \delta_{ij} = \begin{cases} 1, & i = j, \\ 0, & i \neq j, \end{cases}$$

when x_j is a node in the mesh with global node number j. The result $\varphi_i(x$ arises because the Lagrange polynomials are constructed to have exac property. The property also implies a convenient interpretation of c_i as t of u at node i. To show this, we expand u in the usual way as $\sum_j c_$ choose $\psi_i = \varphi_i$:



igure 18: Illustration of the piecewise quadratic basis functions associated ith nodes in element 1.

$$u(x_i) = \sum_{j\in\mathcal{I}_s} c_j\psi_j(x_i) = \sum_{j\in\mathcal{I}_s} c_j\varphi_j(x_i) = c_i\varphi_i(x_i) = c_i\,.$$

ecause of this interpretation, the coefficient c_i is by many named u_i or U_i.

Second, $\varphi_i(x)$ is mostly zero throughout the domain:

- $\varphi_i(x) \neq 0$ only on those elements that contain global node i,
- $\varphi_i(x)\varphi_j(x) \neq 0$ if and only if i and j are global node numbers in the same element.

ince $A_{i,j}$ is the integral of $\varphi_i\varphi_j$ it means that *most of the elements in the oefficient matrix will be zero*. We will come back to these properties and use iem actively in computations to save memory and CPU time.

We let each element have $d+1$ nodes, resulting in local Lagrange polynomials f degree d. It is not a requirement to have the same d value in each element, ut for now we will assume so.

.3 Example on piecewise quadratic finite element functions

igure 18 illustrates how piecewise quadratic basis functions can look like ($d = 2$). /e work with the domain $\Omega = [0, 1]$ divided into four equal-sized elements, each

having three nodes. The `nodes` and `elements` lists in this particular e become

```
nodes = [0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0]
elements = [[0, 1, 2], [2, 3, 4], [4, 5, 6], [6, 7, 8]]
```

Figure 19 sketches the mesh and the numbering. Nodes are marked witl on the x axis and element boundaries are marked with vertical dashed Figure 18.

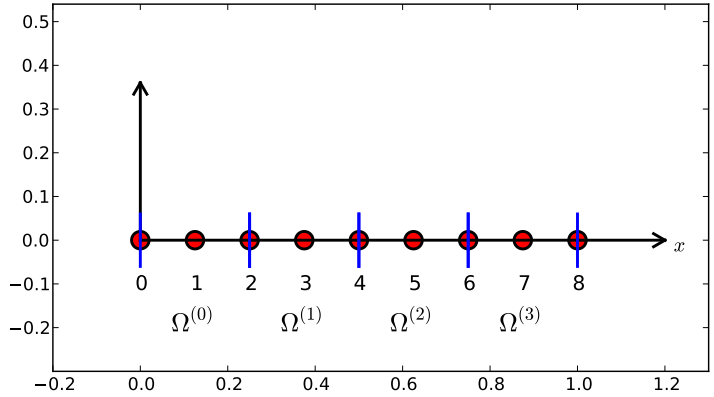


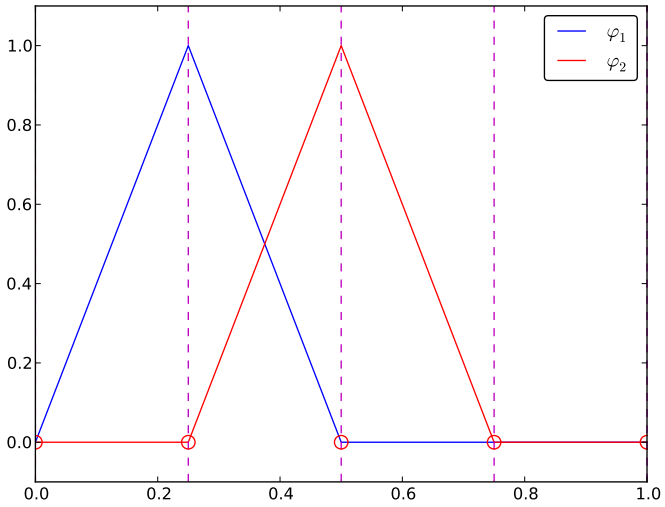
Figure 19: Sketch of mesh with 4 elements and 3 nodes per eleme

Let us explain in detail how the basis functions are constructed ac to the principles. Consider element number 1 in Figure 18, $\Omega^{(1)} = [0$ with local nodes 0, 1, and 2 corresponding to global nodes 2, 3, and coordinates of these nodes are 0.25, 0.375, and 0.5, respectively. We defi Lagrange polynomials on this element:

1. The polynomial that is 1 at local node 1 ($x = 0.375$, global node 3 up the basis function $\varphi_3(x)$ over this element, with $\varphi_3(x) = 0$ out element.
2. The Lagrange polynomial that is 1 at local node 0 is the "rig of the global basis function $\varphi_2(x)$. The "left part" of $\varphi_2(x)$ co a Lagrange polynomial associated with local node 2 in the neig element $\Omega^{(0)} = [0, 0.25]$.
3. Finally, the polynomial that is 1 at local node 2 (global node 4) is part" of the global basis function $\varphi_4(x)$. The "right part" comes f Lagrange polynomial that is 1 at local node 0 in the neighboring $\Omega^{(2)} = [0.5, 0.75]$.

As mentioned earlier, any global basis function $\varphi_i(x)$ is zero on eleme do not contain the node with global node number i.

The other global functions associated with internal nodes, φ_1, φ_5, and φ_7, are ll of the same shape as the drawn φ_3, while the global basis functions associated ith shared nodes also have the same shape, provided the elements are of the ame length.



igure 20: Illustration of the piecewise linear basis functions associated with odes in element 1.

.4 Example on piecewise linear finite element functions

igure 20 shows piecewise linear basis functions ($d = 1$). Also here we have our elements on $\Omega = [0, 1]$. Consider the element $\Omega^{(1)} = [0.25, 0.5]$. Now there re no internal nodes in the elements so that all basis functions are associated ith nodes at the element boundaries and hence made up of two Lagrange olynomials from neighboring elements. For example, $\varphi_1(x)$ results from the agrange polynomial in element 0 that is 1 at local node 1 and 0 at local node , combined with the Lagrange polynomial in element 1 that is 1 at local node 0 nd 0 at local node 1. The other basis functions are constructed similarly.

Explicit mathematical formulas are needed for $\varphi_i(x)$ in computations. In the iecewise linear case, one can show that

$$\varphi_i(x) = \begin{cases} 0, & x < x_{i-1}, \\ (x - x_{i-1})/(x_i - x_{i-1}), & x_{i-1} \leq x < x_i, \\ 1 - (x - x_i)/(x_{i+1} - x_i), & x_i \leq x < x_{i+1}, \\ 0, & x \geq x_{i+1}\,. \end{cases} \tag{53}$$

Here, x_j, $j = i-1, i, i+1$, denotes the coordinate of node j. For elen equal length h the formulas can be simplified to

$$\varphi_i(x) = \begin{cases} 0, & x < x_{i-1}, \\ (x - x_{i-1})/h, & x_{i-1} \leq x < x_i, \\ 1 - (x - x_i)/h, & x_i \leq x < x_{i+1}, \\ 0, & x \geq x_{i+1} \end{cases}$$

3.5 Example on piecewise cubic finite element basis tions

Piecewise cubic basis functions can be defined by introducing four nc element. Figure 21 shows examples on $\varphi_i(x)$, $i = 3, 4, 5, 6$, associat element number 1. Note that φ_4 and φ_5 are nonzero on element number φ_3 and φ_6 are made up of Lagrange polynomials on two neighboring el

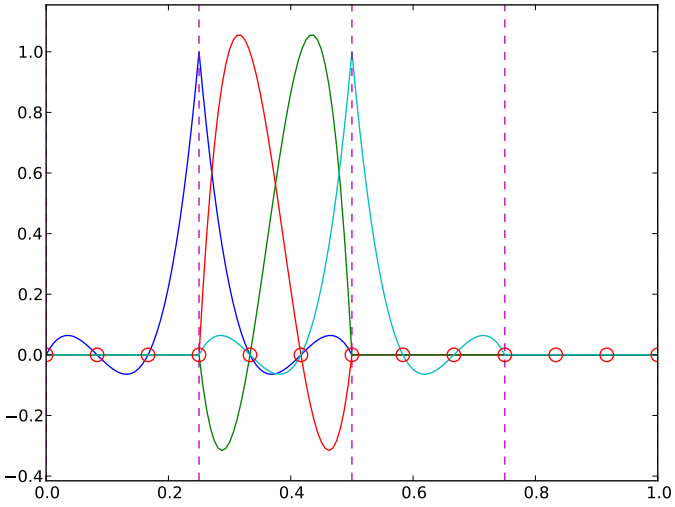


Figure 21: Illustration of the piecewise cubic basis functions associat nodes in element 1.

We see that all the piecewise linear basis functions have the same "hat They are naturally referred to as *hat functions*, also called *chapeau fu* The piecewise quadratic functions in Figure 18 are seen to be of tw "Rounded hats" associated with internal nodes in the elements and son "sombrero" shaped hats associated with element boundary nodes. High basis functions also have hat-like shapes, but the functions have pror oscillations in addition, as illustrated in Figure 21.

A common terminology is to speak about *linear elements* as elements with two ocal nodes associated with piecewise linear basis functions. Similarly, *quadratic lements* and *cubic elements* refer to piecewise quadratic or cubic functions ver elements with three or four local nodes, respectively. Alternative names, equently used later, are P1 elements for linear elements, P2 for quadratic ements, and so forth: Pd signifies degree d of the polynomial basis functions.

.6 Calculating the linear system

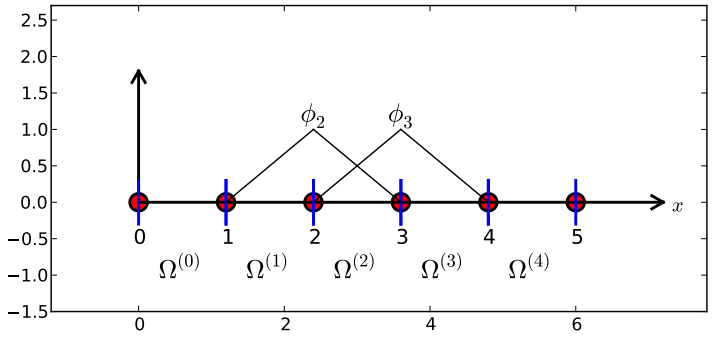
he elements in the coefficient matrix and right-hand side are given by the ormulas (27) and (28), but now the choice of ψ_i is φ_i. Consider P1 elements here $\varphi_i(x)$ piecewise linear. Nodes and elements numbered consecutively from ft to right in a uniformly partitioned mesh imply the nodes

$$x_i = ih, \quad i = 0, \ldots, N,$$

nd the elements

$$\Omega^{(i)} = [x_i, x_{i+1}] = [ih, (i+1)h], \quad i = 0, \ldots, N_e = N - 1\,. \tag{55}$$

Ve have in this case N elements and $N+1$ nodes, and $\Omega = [x_0, x_N]$. The ormula for $\varphi_i(x)$ is given by (54) and a graphical illustration is provided in igures 20 and 23. First we clearly see from the figures the very important roperty $\varphi_i(x)\varphi_j(x) \neq 0$ if and only if $j = i-1$, $j = i$, or $j = i+1$, or lternatively expressed, if and only if i and j are nodes in the same element. otherwise, φ_i and φ_j are too distant to have an overlap and consequently their roduct vanishes.



igure 22: Illustration of the piecewise linear basis functions corresponding to lobal node 2 and 3.

alculating a specific matrix entry. Let us calculate the specific matrix ntry $A_{2,3} = \int_\Omega \varphi_2\varphi_3\,\mathrm{d}x$. Figure 22 shows how φ_2 and φ_3 look like. We realize om this figure that the product $\varphi_2\varphi_3 \neq 0$ only over element 2, which contains

node 2 and 3. The particular formulas for $\varphi_2(x)$ and $\varphi_3(x)$ on $[x_2, x_3]$ a from (54). The function φ_3 has positive slope over $[x_2, x_3]$ and corresp the interval $[x_{i-1}, x_i]$ in (54). With $i = 3$ we get

$$\varphi_3(x) = (x - x_2)/h,$$

while $\varphi_2(x)$ has negative slope over $[x_2, x_3]$ and corresponds to setting (54),

$$\varphi_2(x) = 1 - (x - x_2)/h\,.$$

We can now easily integrate,

$$A_{2,3} = \int_\Omega \varphi_2\varphi_3\,\mathrm{d}x = \int_{x_2}^{x_3} \left(1 - \frac{x - x_2}{h}\right)\frac{x - x_2}{h}\,\mathrm{d}x = \frac{h}{6}\,.$$

The diagonal entry in the coefficient matrix becomes

$$A_{2,2} = \int_{x_1}^{x_2} \left(\frac{x - x_1}{h}\right)^2 \mathrm{d}x + \int_{x_2}^{x_3} \left(1 - \frac{x - x_2}{h}\right)^2 \mathrm{d}x = \frac{2h}{3}\,.$$

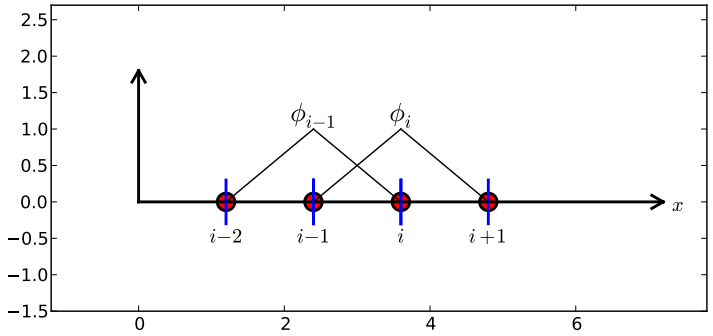
The entry $A_{2,1}$ has an the integral that is geometrically similar to the s in Figure 22, so we get $A_{2,1} = h/6$.

Calculating a general row in the matrix. We can now genera calculation of matrix entries to a general row number i. The entry A $\int_\Omega \varphi_i\varphi_{i-1}\,\mathrm{d}x$ involves hat functions as depicted in Figure 23. Since the is geometrically identical to the situation with specific nodes 2 and 3, w that $A_{i,i-1} = A_{i,i+1} = h/6$ and $A_{i,i} = 2h/3$. However, we can comp integral directly too:

$$\begin{aligned} A_{i,i-1} &= \int_\Omega \varphi_i\varphi_{i-1}\,\mathrm{d}x \\ &= \underbrace{\int_{x_{i-2}}^{x_{i-1}} \varphi_i\varphi_{i-1}\,\mathrm{d}x}_{\varphi_i = 0} + \int_{x_{i-1}}^{x_i} \varphi_i\varphi_{i-1}\,\mathrm{d}x + \underbrace{\int_{x_i}^{x_{i+1}} \varphi_i\varphi_{i-1}\,\mathrm{d}x}_{\varphi_{i-1} = 0} \\ &= \int_{x_{i-1}}^{x_i} \underbrace{\left(\frac{x - x_i}{h}\right)}_{\varphi_i(x)} \underbrace{\left(1 - \frac{x - x_{i-1}}{h}\right)}_{\varphi_{i-1}(x)}\,\mathrm{d}x = \frac{h}{6}\,. \end{aligned}$$

The particular formulas for $\varphi_{i-1}(x)$ and $\varphi_i(x)$ on $[x_{i-1}, x_i]$ are four (54): φ_i is the linear function with positive slope, corresponding to the $[x_{i-1}, x_i]$ in (54), while ϕ_{i-1} has a negative slope so the definition in $[x_i, x_{i+1}]$ in (54) must be used. (The appearance of i in (54) and the might be confusing, as we speak about two different i indices.)

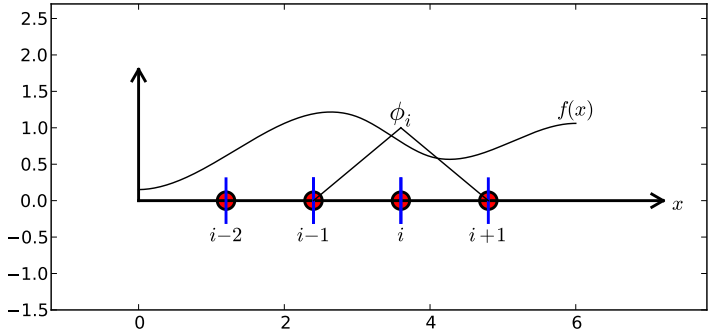
The first and last row of the coefficient matrix lead to slightly c integrals:



igure 23: Illustration of two neighboring linear (hat) functions with general ode numbers.

$$A_{0,0} = \int_\Omega \varphi_0^2 \,\mathrm{d}x = \int_{x_0}^{x_1} \left(1 - \frac{x - x_0}{h}\right)^2 \mathrm{d}x = \frac{h}{3}\,.$$

imilarly, $A_{N,N}$ involves an integral over only one element and equals hence $h/3$.



igure 24: Right-hand side integral with the product of a basis function and ıe given function to approximate.

The general formula for b_i, see Figure 24, is now easy to set up

$$_i = \int_\Omega \varphi_i(x) f(x)\,\mathrm{d}x = \int_{x_{i-1}}^{x_i} \frac{x - x_{i-1}}{h} f(x)\,\mathrm{d}x + \int_{x_i}^{x_{i+1}} \left(1 - \frac{x - x_i}{h}\right) f(x)\,\mathrm{d}x\,. \tag{56}$$

/e need a specific $f(x)$ function to compute these integrals. With two equal-sized .ements in $\Omega = [0, 1]$ and $f(x) = x(1 - x)$, one gets

$$A = \frac{h}{6}\begin{pmatrix} 2 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 2 \end{pmatrix}, \quad b = \frac{h^2}{12}\begin{pmatrix} 2 - 3h \\ 12 - 14h \\ 10 - 17h \end{pmatrix}\,.$$

The solution becomes

$$c_0 = \frac{h^2}{6}, \quad c_1 = h - \frac{5}{6}h^2, \quad c_2 = 2h - \frac{23}{6}h^2\,.$$

The resulting function

$$u(x) = c_0\varphi_0(x) + c_1\varphi_1(x) + c_2\varphi_2(x)$$

is displayed in Figure 25 (left). Doubling the number of elements to fo to the improved approximation in the right part of Figure 25.

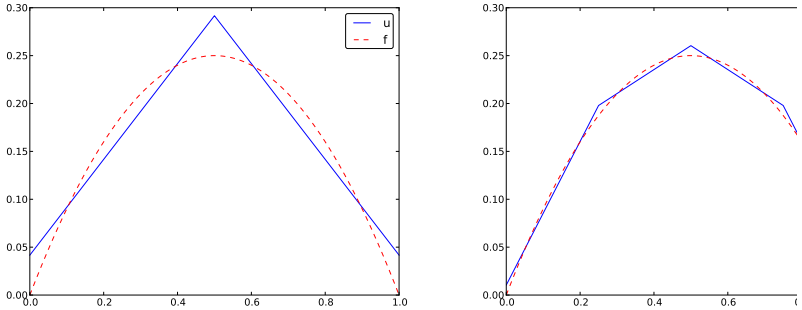


Figure 25: Least squares approximation of a parabola using 2 (left (right) P1 elements.

3.7 Assembly of elementwise computations

The integrals above are naturally split into integrals over individual e since the formulas change with the elements. This idea of splitting the is fundamental in all practical implementations of the finite element me

Let us split the integral over Ω into a sum of contributions from each

$$A_{i,j} = \int_\Omega \varphi_i\varphi_j\,\mathrm{d}x = \sum_e A_{i,j}^{(e)}, \quad A_{i,j}^{(e)} = \int_{\Omega^{(e)}} \varphi_i\varphi_j\,\mathrm{d}x\,.$$

Now, $A_{i,j}^{(e)} \neq 0$ if and only if i and j are nodes in element e (look at Figu realize this property). Introduce $i = q(e, r)$ as the mapping of local node r in element e to the global node number i. This is just a short mathe notation for the expression `i=elements[e][r]` in a program. Let r a the local node numbers corresponding to the global node numbers i = and $j = q(e, s)$. With d nodes per element, all the nonzero elements in A from the integrals involving basis functions with indices correspondin global node numbers in element number e:

$$\int_{\Omega^{(e)}} \varphi_{q(e,r)}\varphi_{q(e,s)}\,\mathrm{d}x, \quad r, s = 0, \ldots, d\,.$$

hese contributions can be collected in a $(d+1)\times(d+1)$ matrix known as the *ement matrix*. Let $I_d = \{0,\ldots,d\}$ be the valid indices of r and s. We introduce ıe notation

$$\tilde{A}^{(e)} = \{\tilde{A}^{(e)}_{r,s}\}, \quad r,s \in I_d,$$

ır the element matrix. For the case $d=2$ we have

$$\tilde{A}^{(e)} = \begin{bmatrix} \tilde{A}^{(e)}_{0,0} & \tilde{A}^{(e)}_{0,1} & \tilde{A}^{(e)}_{0,2} \\ \tilde{A}^{(e)}_{1,0} & \tilde{A}^{(e)}_{1,1} & \tilde{A}^{(e)}_{1,2} \\ \tilde{A}^{(e)}_{2,0} & \tilde{A}^{(e)}_{2,1} & \tilde{A}^{(e)}_{2,2} \end{bmatrix}.$$

iiven the numbers $\tilde{A}^{(e)}_{r,s}$, we should according to (57) add the contributions to ıe global coefficient matrix by

$$A_{q(e,r),q(e,s)} := A_{q(e,r),q(e,s)} + \tilde{A}^{(e)}_{r,s}, \quad r,s \in I_d\,. \tag{58}$$

his process of adding in elementwise contributions to the global matrix is called *nite element assembly* or simply *assembly*. Figure 26 illustrates how element ıatrices for elements with two nodes are added into the global matrix. More ɔecifically, the figure shows how the element matrix associated with elements 1 nd 2 assembled, assuming that global nodes are numbered from left to right in ıe domain. With regularly numbered P3 elements, where the element matrices ave size 4×4, the assembly of elements 1 and 2 are sketched in Figure 27.

After assembly of element matrices corresponding to regularly numbered ements and nodes are understood, it is wise to study the assembly process for regularly numbered elements and nodes. Figure 17 shows a mesh where the `elements` array, or $q(e,r)$ mapping in mathematical notation, is given as

```
elements = [[2, 1], [4, 5], [0, 4], [3, 0], [5, 2]]
```

he associated assembly of element matrices 1 and 2 is sketched in Figure 28.

These three assembly processes can also be animated[6].

The right-hand side of the linear system is also computed elementwise:

$$b_i = \int_\Omega f(x)\varphi_i(x)\,\mathrm{d}x = \sum_e b_i^{(e)}, \quad b_i^{(e)} = \int_{\Omega^{(e)}} f(x)\varphi_i(x)\,\mathrm{d}x\,. \tag{59}$$

/e observe that $b_i^{(e)} \neq 0$ if and only if global node i is a node in element e (look t Figure 24 to realize this property). With d nodes per element we can collect ıe $d+1$ nonzero contributions $b_i^{(e)}$, for $i = q(e,r)$, $r \in I_d$, in an *element vector*

$$\tilde{b}^{(e)}_r = \{\tilde{b}^{(e)}_r\}, \quad r \in I_d\,.$$

hese contributions are added to the global right-hand side by an assembly rocess similar to that for the element matrices:

$$b_{q(e,r)} := b_{q(e,r)} + \tilde{b}^{(e)}_r, \quad r \in I_d\,. \tag{60}$$

[6] `http://tinyurl.com/opdfafk/pub/mov-fem/fe_assembly.html`

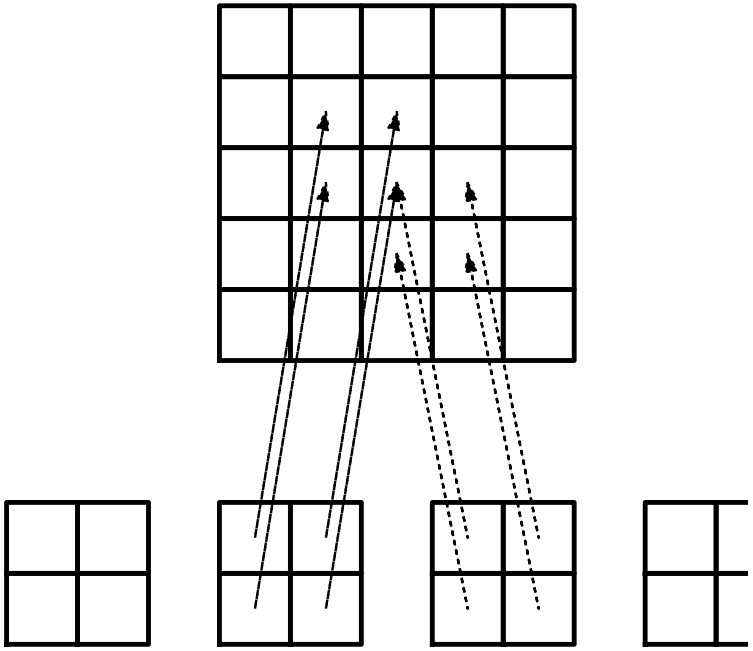


Figure 26: Illustration of matrix assembly: regularly numbered P1 ele

3.8 Mapping to a reference element

Instead of computing the integrals

$$\tilde{A}^{(e)}_{r,s} = \int_{\Omega^{(e)}} \varphi_{q(e,r)}(x)\varphi_{q(e,s)}(x)\,\mathrm{d}x$$

over some element $\Omega^{(e)} = [x_L, x_R]$, it is convenient to map the element $[x_L, x_R]$ to a standardized reference element domain $[-1,1]$. (We ha introduced x_L and x_R as the left and right boundary points of an a element. With a natural, regular numbering of nodes and elements fron right through the domain, we have $x_L = x_e$ and $x_R = x_{e+1}$ for P1 elem

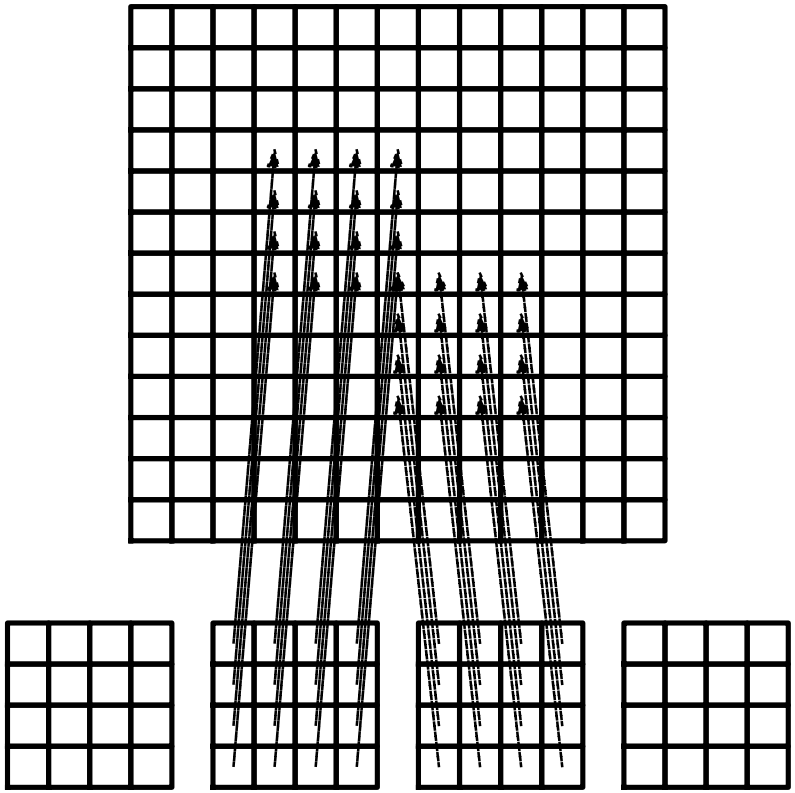
Let $X \in [-1,1]$ be the coordinate in the reference element. A linear *mapping* from X to x reads

$$x = \frac{1}{2}(x_L + x_R) + \frac{1}{2}(x_R - x_L)X\,.$$

This relation can alternatively be expressed by

$$x = x_m + \frac{1}{2}hX,$$

where we have introduced the element midpoint $x_m = (x_L + x_R)/2$ element length $h = x_R - x_L$.



igure 27: Illustration of matrix assembly: regularly numbered P3 elements.

Integrating on the reference element is a matter of just changing the integra-on variable from x to X. Let

$$\tilde{\varphi}_r(X) = \varphi_{q(e,r)}(x(X)) \tag{63}$$

e the basis function associated with local node number r in the reference ement. The integral transformation reads

$$\tilde{A}^{(e)}_{r,s} = \int_{\Omega^{(e)}} \varphi_{q(e,r)}(x)\varphi_{q(e,s)}(x)\,\mathrm{d}x = \int_{-1}^{1} \tilde{\varphi}_r(X)\tilde{\varphi}_s(X)\frac{dx}{dX}\,\mathrm{d}X\,. \tag{64}$$

he stretch factor dx/dX between the x and X coordinates becomes the deter-inant of the Jacobian matrix of the mapping between the coordinate systems ı 2D and 3D. To obtain a uniform notation for 1D, 2D, and 3D problems we nerefore replace dx/dX by $\det J$ already now. In 1D, $\det J = dx/dX = h/2$. he integration over the reference element is then written as

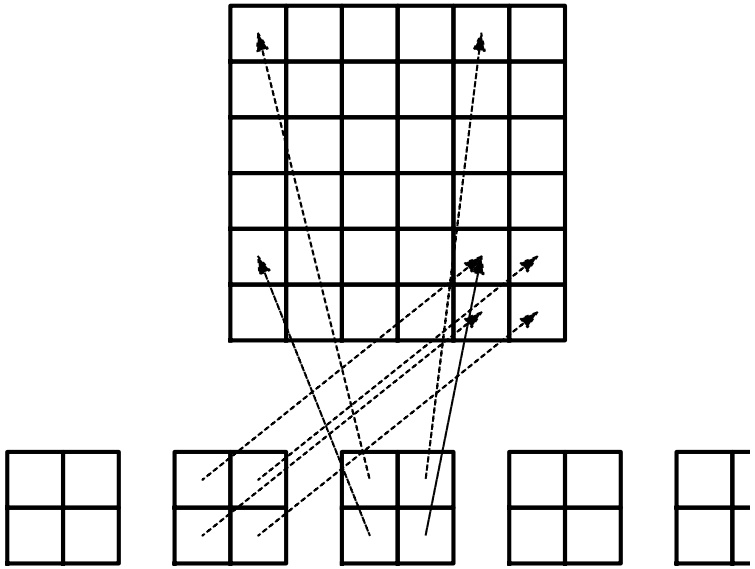


Figure 28: Illustration of matrix assembly: irregularly numbered P1 ele

$$\tilde{A}^{(e)}_{r,s} = \int_{-1}^{1} \tilde{\varphi}_r(X)\tilde{\varphi}_s(X)\det J\,dX\,.$$

The corresponding formula for the element vector entries becomes

$$\tilde{b}^{(e)}_{r} = \int_{\Omega^{(e)}} f(x)\varphi_{q(e,r)}(x)dx = \int_{-1}^{1} f(x(X))\tilde{\varphi}_r(X)\det J\,dX\,.$$

Since we from now on will work in the reference element, we need mathematical formulas for the basis functions $\varphi_i(x)$ in the reference only, i.e., we only need to specify formulas for $\tilde{\varphi}_r(X)$. This is a very co simplification compared to specifying piecewise polynomials in the p domain.

The $\tilde{\varphi}_r(x)$ functions are simply the Lagrange polynomials defined t the local nodes in the reference element. For $d = 1$ and two nodes per e we have the linear Lagrange polynomials

$$\tilde{\varphi}_0(X) = \frac{1}{2}(1 - X)$$
$$\tilde{\varphi}_1(X) = \frac{1}{2}(1 + X)$$

Quadratic polynomials, $d = 2$, have the formulas

$$\tilde{\varphi}_0(X) = \frac{1}{2}(X-1)X \tag{69}$$

$$\tilde{\varphi}_1(X) = 1 - X^2 \tag{70}$$

$$\tilde{\varphi}_2(X) = \frac{1}{2}(X+1)X \tag{71}$$

ı general,

$$\tilde{\varphi}_r(X) = \prod_{s=0, s\neq r}^{d} \frac{X - X_{(s)}}{X_{(r)} - X_{(s)}}, \tag{72}$$

here $X_{(0)}, \ldots, X_{(d)}$ are the coordinates of the local nodes in the reference ement. These are normally uniformly spaced: $X_{(r)} = -1 + 2r/d$, $r \in I_d$.

Why reference elements?

The great advantage of using reference elements is that the formulas for the basis functions, $\tilde{\varphi}_r(X)$, are the same for all elements and independent of the element geometry (length and location in the mesh). The geometric information is "factored out" in the simple mapping formula and the associated $\det J$ quantity, but this information is (here taken as) the same for element types. Also, the integration domain is the same for all elements.

.9 Example: Integration over a reference element

o illustrate the concepts from the previous section in a specific example, we ow consider calculation of the element matrix and vector for a specific choice f d and $f(x)$. A simple choice is $d = 1$ (P1 elements) and $f(x) = x(1-x)$ n $\Omega = [0, 1]$. We have the general expressions (65) and (66) for $\tilde{A}^{(e)}_{r,s}$ and $\tilde{b}^{(e)}_r$. /riting these out for the choices (67) and (68), and using that $\det J = h/2$, we an do the following calculations of the element matrix entries:

$$\begin{aligned}
\tilde{A}^{(e)}_{0,0} &= \int_{-1}^{1} \tilde{\varphi}_0(X)\tilde{\varphi}_0(X)\frac{h}{2}dX \\
&= \int_{-1}^{1} \frac{1}{2}(1-X)\frac{1}{2}(1-X)\frac{h}{2}dX = \frac{h}{8}\int_{-1}^{1}(1-X)^2dX = \frac{h}{3}, \\
\tilde{A}^{(e)}_{1,0} &= \int_{-1}^{1} \tilde{\varphi}_1(X)\tilde{\varphi}_0(X)\frac{h}{2}dX \\
&= \int_{-1}^{1} \frac{1}{2}(1+X)\frac{1}{2}(1-X)\frac{h}{2}dX = \frac{h}{8}\int_{-1}^{1}(1-X^2)dX = \frac{h}{6}, \\
\tilde{A}^{(e)}_{0,1} &= \tilde{A}^{(e)}_{1,0}, \\
\tilde{A}^{(e)}_{1,1} &= \int_{-1}^{1} \tilde{\varphi}_1(X)\tilde{\varphi}_1(X)\frac{h}{2}dX \\
&= \int_{-1}^{1} \frac{1}{2}(1+X)\frac{1}{2}(1+X)\frac{h}{2}dX = \frac{h}{8}\int_{-1}^{1}(1+X)^2dX = \frac{h}{3}\,.
\end{aligned}$$

The corresponding entries in the element vector becomes

$$\begin{aligned}
\tilde{b}^{(e)}_0 &= \int_{-1}^{1} f(x(X))\tilde{\varphi}_0(X)\frac{h}{2}dX \\
&= \int_{-1}^{1} (x_m + \frac{1}{2}hX)(1 - (x_m + \frac{1}{2}hX))\frac{1}{2}(1-X)\frac{h}{2}dX \\
&= -\frac{1}{24}h^3 + \frac{1}{6}h^2x_m - \frac{1}{12}h^2 - \frac{1}{2}hx_m^2 + \frac{1}{2}hx_m \\
\tilde{b}^{(e)}_1 &= \int_{-1}^{1} f(x(X))\tilde{\varphi}_1(X)\frac{h}{2}dX \\
&= \int_{-1}^{1} (x_m + \frac{1}{2}hX)(1 - (x_m + \frac{1}{2}hX))\frac{1}{2}(1+X)\frac{h}{2}dX \\
&= -\frac{1}{24}h^3 - \frac{1}{6}h^2x_m + \frac{1}{12}h^2 - \frac{1}{2}hx_m^2 + \frac{1}{2}hx_m\,.
\end{aligned}$$

In the last two expressions we have used the element midpoint x_m.

Integration of lower-degree polynomials above is tedious, and highe polynomials involve very much more algebra, but `sympy` may help. For e we can easily calculate (73), (73), and (77) by

```
>>> import sympy as sp
>>> x, x_m, h, X = sp.symbols('x x_m h X')
>>> sp.integrate(h/8*(1-X)**2, (X, -1, 1))
h/3
>>> sp.integrate(h/8*(1+X)*(1-X), (X, -1, 1))
h/6
>>> x = x_m + h/2*X
>>> b_0 = sp.integrate(h/4*x*(1-x)*(1-X), (X, -1, 1))
```

```
>>> print b_0
-h**3/24 + h**2*x_m/6 - h**2/12 - h*x_m**2/2 + h*x_m/2
```

or inclusion of formulas in documents (like the present one), `sympy` can print xpressions in LaTeX format:

```
>>> print sp.latex(b_0, mode='plain')
- \frac{1}{24} h^{3} + \frac{1}{6} h^{2} x_{m}
- \frac{1}{12} h^{2} - \half h x_{m}^{2}
+ \half h x_{m}
```

Implementation

ased on the experience from the previous example, it makes sense to write ome code to automate the analytical integration process for any choice of finite lement basis functions. In addition, we can automate the assembly process nd linear system solution. Appropriate functions for this purpose document ll details of all steps in the finite element computations and can found in the nodule file `fe_approx1D.py`[7]. The key steps in the computational machinery re now explained in detail in terms of code and text.

.1 Integration

irst we need a Python function for defining $\tilde{\varphi}_r(X)$ in terms of a Lagrange olynomial of degree `d`:

```
import sympy as sp
import numpy as np

def phi_r(r, X, d):
    if isinstance(X, sp.Symbol):
        h = sp.Rational(1, d)  # node spacing
        nodes = [2*i*h - 1 for i in range(d+1)]
    else:
        # assume X is numeric: use floats for nodes
        nodes = np.linspace(-1, 1, d+1)
    return Lagrange_polynomial(X, r, nodes)

def Lagrange_polynomial(x, i, points):
    p = 1
    for k in range(len(points)):
        if k != i:
            p *= (x - points[k])/(points[i] - points[k])
    return p
```

bserve how we construct the `phi_r` function to be a symbolic expression for $_r(X)$ if `X` is a `Symbol` object from `sympy`. Otherwise, we assume that `X` is a `float` bject and compute the corresponding floating-point value of $\tilde{\varphi}_r(X)$. Recall that

[7] http://tinyurl.com/nm5587k/fem/fe_approx1D.py

the `Lagrange_polynomial` function, here simply copied from Section 2.' with both symbolic and numeric variables.

The complete basis $\tilde{\varphi}_0(X), \ldots, \tilde{\varphi}_d(X)$ on the reference element, repr as a list of symbolic expressions, is constructed by

```
def basis(d=1):
    X = sp.Symbol('X')
    phi = [phi_r(r, X, d) for r in range(d+1)]
    return phi
```

Now we are in a position to write the function for computing the element

```
def element_matrix(phi, Omega_e, symbolic=True):
    n = len(phi)
    A_e = sp.zeros((n, n))
    X = sp.Symbol('X')
    if symbolic:
        h = sp.Symbol('h')
    else:
        h = Omega_e[1] - Omega_e[0]
    detJ = h/2  # dx/dX
    for r in range(n):
        for s in range(r, n):
            A_e[r,s] = sp.integrate(phi[r]*phi[s]*detJ, (X, -1, 1
            A_e[s,r] = A_e[r,s]
    return A_e
```

In the symbolic case (`symbolic` is `True`), we introduce the element le a symbol `h` in the computations. Otherwise, the real numerical valu element interval `Omega_e` is used and the final matrix elements are numb symbols. This functionality can be demonstrated:

```
>>> from fe_approx1D import *
>>> phi = basis(d=1)
>>> phi
[1/2 - X/2, 1/2 + X/2]
>>> element_matrix(phi, Omega_e=[0.1, 0.2], symbolic=True)
[h/3, h/6]
[h/6, h/3]
>>> element_matrix(phi, Omega_e=[0.1, 0.2], symbolic=False)
[0.0333333333333333, 0.0166666666666667]
[0.0166666666666667, 0.0333333333333333]
```

The computation of the element vector is done by a similar procedu

```
def element_vector(f, phi, Omega_e, symbolic=True):
    n = len(phi)
    b_e = sp.zeros((n, 1))
    # Make f a function of X
    X = sp.Symbol('X')
    if symbolic:
        h = sp.Symbol('h')
    else:
        h = Omega_e[1] - Omega_e[0]
    x = (Omega_e[0] + Omega_e[1])/2 + h/2*X  # mapping
```

```
    f = f.subs('x', x)  # substitute mapping formula for x
    detJ = h/2  # dx/dX
    for r in range(n):
        b_e[r] = sp.integrate(f*phi[r]*detJ, (X, -1, 1))
    return b_e
```

ere we need to replace the symbol `x` in the expression for `f` by the mapping ormula such that `f` can be integrated in terms of X, cf. the formula $\tilde{b}_r^{(e)} = \int_{-1}^{1} f(x(X))\tilde{\varphi}_r(X)\frac{h}{2}dX$.

The integration in the element matrix function involves only products of olynomials, which `sympy` can easily deal with, but for the right-hand side `sympy` ıay face difficulties with certain types of expressions `f`. The result of the integral then an `Integral` object and not a number or expression as when symbolic ıtegration is successful. It may therefore be wise to introduce a fallback on umerical integration. The symbolic integration can also take much time before n unsuccessful conclusion so we may also introduce a parameter `symbolic` and et it to `False` to avoid symbolic integration:

```
lef element_vector(f, phi, Omega_e, symbolic=True):
        ...
        if symbolic:
            I = sp.integrate(f*phi[r]*detJ, (X, -1, 1))
        if not symbolic or isinstance(I, sp.Integral):
            h = Omega_e[1] - Omega_e[0]  # Ensure h is numerical
            detJ = h/2
            integrand = sp.lambdify([X], f*phi[r]*detJ)
            I = sp.mpmath.quad(integrand, [-1, 1])
        b_e[r] = I
        ...
```

umerical integration requires that the symbolic integrand is converted to a plain ython function (`integrand`) and that the element length `h` is a real number.

.2 Linear system assembly and solution

he complete algorithm for computing and assembling the elementwise contribuons takes the following form

```
lef assemble(nodes, elements, phi, f, symbolic=True):
    N_n, N_e = len(nodes), len(elements)
    if symbolic:
        A = sp.zeros((N_n, N_n))
        b = sp.zeros((N_n, 1))    # note: (N_n, 1) matrix
    else:
        A = np.zeros((N_n, N_n))
        b = np.zeros(N_n)
    for e in range(N_e):
        Omega_e = [nodes[elements[e][0]], nodes[elements[e][-1]]]

        A_e = element_matrix(phi, Omega_e, symbolic)
        b_e = element_vector(f, phi, Omega_e, symbolic)

        for r in range(len(elements[e])):
```



```
            for s in range(len(elements[e])):
                A[elements[e][r],elements[e][s]] += A_e[r,s]
            b[elements[e][r]] += b_e[r]
    return A, b
```

The `nodes` and `elements` variables represent the finite element mesh as ex earlier.

Given the coefficient matrix `A` and the right-hand side `b`, we can c the coefficients $\{c_i\}_{i\in\mathcal{I}_s}$ in the expansion $u(x) = \sum_j c_j\varphi_j$ as the solutio `c` of the linear system:

```
if symbolic:
    c = A.LUsolve(b)
else:
    c = np.linalg.solve(A, b)
```

When `A` and `b` are `sympy` arrays, the solution procedure implied by `A.LU` symbolic. Otherwise, `A` and `b` are `numpy` arrays and a standard numerica is called. The symbolic version is suited for small problems only (small N since the calculation time becomes prohibitively large otherwise. Norma symbolic *integration* will be more time consuming in small problems t symbolic *solution* of the linear system.

4.3 Example on computing symbolic approximation

We can exemplify the use of `assemble` on the computational case from Sec with two P1 elements (linear basis functions) on the domain $\Omega = [0, 1]$ first work with a symbolic element length:

```
>>> h, x = sp.symbols('h x')
>>> nodes = [0, h, 2*h]
>>> elements = [[0, 1], [1, 2]]
>>> phi = basis(d=1)
>>> f = x*(1-x)
>>> A, b = assemble(nodes, elements, phi, f, symbolic=True)
>>> A
[h/3,   h/6,   0]
[h/6, 2*h/3, h/6]
[  0,   h/6, h/3]
>>> b
[     h**2/6 - h**3/12]
[      h**2 - 7*h**3/6]
[5*h**2/6 - 17*h**3/12]
>>> c = A.LUsolve(b)
>>> c
[                              h**2/6]
[12*(7*h**2/12 - 35*h**3/72)/(7*h)]
[  7*(4*h**2/7 - 23*h**3/21)/(2*h)]
```

.4 Comparison with finite elements and interpolation/-collocation

/e may, for comparison, compute the `c` vector corresponding to an interpola-on/collocation method with finite element basis functions. Choosing the nodes s points, the principle is

$$u(x_i) = \sum_{j \in \mathcal{I}_s} c_j \varphi_j(x_i) = f(x_i), \quad i \in \mathcal{I}_s\,.$$

he coefficient matrix $A_{i,j} = \varphi_j(x_i)$ becomes the identity matrix because basis unction number j vanishes at all nodes, except node j: $\varphi_j(x_i = \delta_{ij}$. Therefore, $= f(x_i$.

The associated `sympy` calculations are

```
>>> fn = sp.lambdify([x], f)
>>> c = [fn(xc) for xc in nodes]
>>> c
[0, h*(1 - h), 2*h*(1 - 2*h)]
```

hese expressions are much simpler than those based on least squares or projec-on in combination with finite element basis functions.

.5 Example on computing numerical approximations

he numerical computations corresponding to the symbolic ones in Section 4.3, nd still done by `sympy` and the `assemble` function, go as follows:

```
>>> nodes = [0, 0.5, 1]
>>> elements = [[0, 1], [1, 2]]
>>> phi = basis(d=1)
>>> x = sp.Symbol('x')
>>> f = x*(1-x)
>>> A, b = assemble(nodes, elements, phi, f, symbolic=False)
>>> A
[ 0.166666666666667, 0.0833333333333333,                   0]
[0.0833333333333333,  0.333333333333333, 0.0833333333333333]
[                 0, 0.0833333333333333,  0.166666666666667]
>>> b
[          0.03125]
[0.104166666666667]
[          0.03125]
>>> c = A.LUsolve(b)
>>> c
[0.0416666666666666]
[ 0.291666666666667]
[0.0416666666666666]
```

The `fe_approx1D` module contains functions for generating the `nodes` and `lements` lists for equal-sized elements with any number of nodes per element. he coordinates in `nodes` can be expressed either through the element length ymbol `h` (`symbolic=True`) or by real numbers (`symbolic=False`):


```
nodes, elements = mesh_uniform(N_e=10, d=3, Omega=[0,1],
                               symbolic=True)
```

There is also a function

```
def approximate(f, symbolic=False, d=1, N_e=4, filename='tmp.pdf'
```

which computes a mesh with `N_e` elements, basis functions of degree approximates a given symbolic expression `f` by a finite element expansion $\sum_j c_j\varphi_j(x)$. When `symbolic` is `False`, $u(x) = \sum_j c_j\varphi_j(x)$ can be comp a (large) number of points and plotted together with $f(x)$. The construct points from the solution vector `c` is done elementwise by evaluating $\sum_r$ at a (large) number of points in each element in the local coordinate and the discrete (x, u) values on each element are stored in separate arr are finally concatenated to form a global array for x and for u. The de found in the `u_glob` function in `fe_approx1D.py`.

4.6 The structure of the coefficient matrix

Let us first see how the global matrix looks like if we assemble symbolic matrices, expressed in terms of `h`, from several elements:

```
>>> d=1; N_e=8; Omega=[0,1]  # 8 linear elements on [0,1]
>>> phi = basis(d)
>>> f = x*(1-x)
>>> nodes, elements = mesh_symbolic(N_e, d, Omega)
>>> A, b = assemble(nodes, elements, phi, f, symbolic=True)
>>> A
[h/3,   h/6,     0,     0,     0,     0,     0,     0,   0]
[h/6, 2*h/3,   h/6,     0,     0,     0,     0,     0,   0]
[  0,   h/6, 2*h/3,   h/6,     0,     0,     0,     0,   0]
[  0,     0,   h/6, 2*h/3,   h/6,     0,     0,     0,   0]
[  0,     0,     0,   h/6, 2*h/3,   h/6,     0,     0,   0]
[  0,     0,     0,     0,   h/6, 2*h/3,   h/6,     0,   0]
[  0,     0,     0,     0,     0,   h/6, 2*h/3,   h/6,   0]
[  0,     0,     0,     0,     0,     0,   h/6, 2*h/3, h/6]
[  0,     0,     0,     0,     0,     0,     0,   h/6, h/3]
```

The reader is encouraged to assemble the element matrices by hand an this result, as this exercise will give a hands-on understanding of w assembly is about. In general we have a coefficient matrix that is tridia

$$A = \frac{h}{6}\begin{pmatrix} 2 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 1 & 4 & 1 & \ddots & & & & & \vdots \\ 0 & 1 & 4 & 1 & \ddots & & & & \vdots \\ \vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & 0 & 1 & 4 & 1 & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & & \ddots & 1 & 4 & 1 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 1 & 2 \end{pmatrix} \tag{79}$$

The structure of the right-hand side is more difficult to reveal since it involves n assembly of elementwise integrals of $f(x(X))\tilde{\varphi}_r(X)h/2$, which obviously epend on the particular choice of $f(x)$. Numerical integration can give some nsight into the nature of the right-hand side. For this purpose it is easier to ook at the integration in x coordinates, which gives the general formula (56). or equal-sized elements of length h, we can apply the Trapezoidal rule at the lobal node points to arrive at

$$b_i = h\left(\frac{1}{2}\varphi_i(x_0)f(x_0) + \frac{1}{2}\varphi_i(x_N)f(x_N) + \sum_{j=1}^{N-1}\varphi_i(x_j)f(x_j)\right) \tag{80}$$

$$= \begin{cases} \frac{1}{2}hf(x_i), & i = 0 \text{ or } i = N, \\ hf(x_i), & 1 \leq i \leq N-1 \end{cases} \tag{81}$$

he reason for this simple formula is simply that φ_i is either 0 or 1 at the nodes nd 0 at all but one of them.

Going to P2 elements (`d=2`) leads to the element matrix

$$A^{(e)} = \frac{h}{30}\begin{pmatrix} 4 & 2 & -1 \\ 2 & 16 & 2 \\ -1 & 2 & 4 \end{pmatrix} \tag{82}$$

nd the following global assembled matrix from four elements:

$$A = \frac{h}{30}\begin{pmatrix} 4 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 16 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & 8 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 16 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & 8 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 16 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & 8 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 16 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & 4 \end{pmatrix} \tag{83}$$

In general, for i odd we have the nonzeroes

$$A_{i,i-2} = -1, \quad A_{i-1,i} = 2, \quad A_{i,i} = 8, \quad A_{i+1,i} = 2, \quad A_{i+2,i} = -1$$

multiplied by $h/30$, and for i even we have the nonzeros

$$A_{i-1,i} = 2, \quad A_{i,i} = 16, \quad A_{i+1,i} = 2,$$

multiplied by $h/30$. The rows with odd numbers correspond to node element boundaries and get contributions from two neighboring element assembly process, while the even numbered rows correspond to internal the elements where the only one element contributes to the values in th matrix.

4.7 Applications

With the aid of the `approximate` function in the `fe_approx1D` module easily investigate the quality of various finite element approximations given functions. Figure 29 shows how linear and quadratic elements appro the polynomial $f(x) = x(1-x)^8$ on $\Omega = [0,1]$, using equal-sized elemer results arise from the program

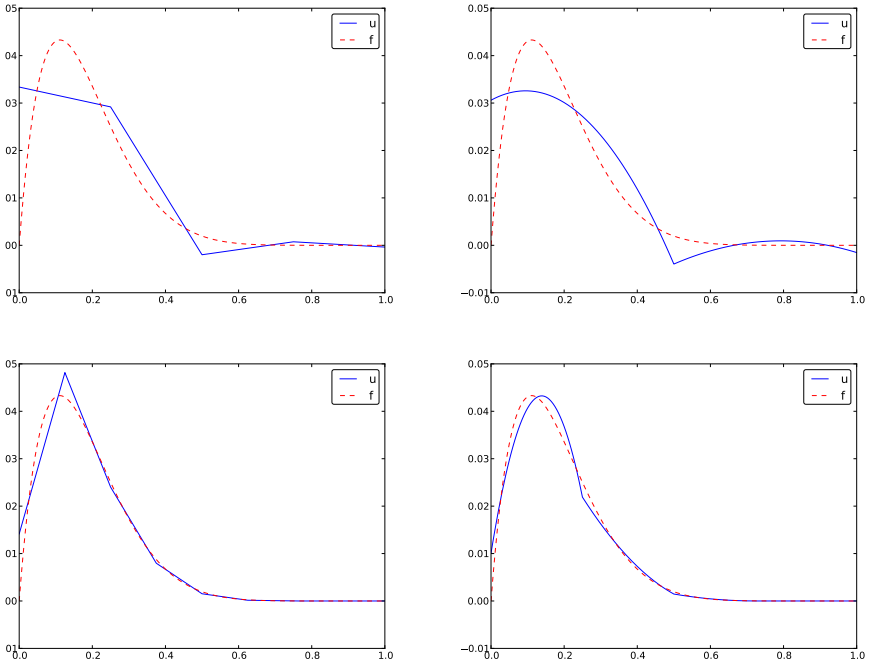
```
import sympy as sp
from fe_approx1D import approximate
x = sp.Symbol('x')

approximate(f=x*(1-x)**8, symbolic=False, d=1, N_e=4)
approximate(f=x*(1-x)**8, symbolic=False, d=2, N_e=2)
approximate(f=x*(1-x)**8, symbolic=False, d=1, N_e=8)
approximate(f=x*(1-x)**8, symbolic=False, d=2, N_e=4)
```

The quadratic functions are seen to be better than the linear ones for t value of N, as we increase N. This observation has some generality: degree is not necessarily better on a coarse mesh, but it is as we refi mesh.

4.8 Sparse matrix storage and solution

Some of the examples in the preceding section took several minutes to c even on small meshes consisting of up to eight elements. The main expl for slow computations is unsuccessful symbolic integration: `sympy` ma lot of energy on integrals like $\int f(x(X))\tilde{\varphi}_r(X)h/2dx$ before giving up, program then resorts to numerical integration. Codes that can deal with number of basis functions and accept flexible choices of $f(x)$ should com integrals numerically and replace the matrix objects from `sympy` by the f efficient array objects from `numpy`.

Another reason for slow code is related to the fact that most of the entries $A_{i,j}$ are zero, because $(\varphi_i, \varphi_j) = 0$ unless i and j are nodes in t



igure 29: Comparison of the finite element approximations: 4 P1 elements with nodes (upper left), 2 P2 elements with 5 nodes (upper right), 8 P1 elements ith 9 nodes (lower left), and 4 P2 elements with 9 nodes (lower right).

ement. A matrix whose majority of entries are zeros, is known as a *sparse* atrix. The sparsity should be utilized in software as it dramatically decreases ne storage demands and the CPU-time needed to compute the solution of the near system. This optimization is not critical in 1D problems where modern mputers can afford computing with all the zeros in the complete square matrix, ut in 2D and especially in 3D, sparse matrices are fundamental for feasible nite element computations.

In 1D problems, using a numbering of nodes and elements from left to right ver the domain, the assembled coefficient matrix has only a few diagonals ifferent from zero. More precisely, $2d+1$ diagonals are different from zero. /ith a different numbering of global nodes, say a random ordering, the diagonal ructure is lost, but the number of nonzero elements is unaltered. Figures 30 nd 31 exemplify sparsity patterns.

The `scipy.sparse` library supports creation of sparse matrices and linear ystem solution.

- `scipy.sparse.diags` for matrix defined via diagonals
- `scipy.sparse.lil_matrix` for creation via setting matrix entries

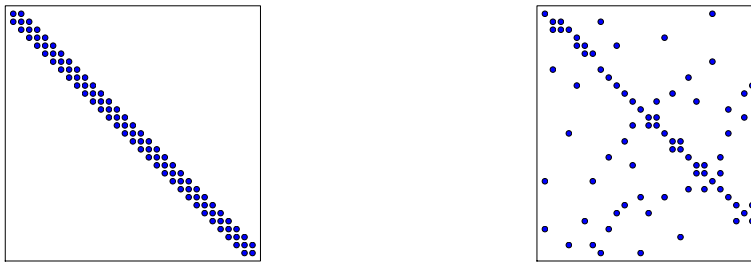


Figure 30: Matrix sparsity pattern for left-to-right numbering (left) and numbering (right) of nodes in P1 elements.

Figure 31: Matrix sparsity pattern for left-to-right numbering (left) and numbering (right) of nodes in P3 elements.

- `scipy.sparse.dok_matrix` for creation via setting matrix entries

5 Comparison of finite element and finite d ence approximation

The previous sections on approximating f by a finite element function the projection/Galerkin or least squares approaches to minimize the mation error. We may, alternatively, use the collocation/interpolation as described in Section 4.4. Here we shall compare these three approacl what one does in the finite difference method when representing a given on a mesh.

5.1 Finite difference approximation of given functic

Approximating a given function $f(x)$ on a mesh in a finite difference cont typically just sample f at the mesh points. If u_i is the value of the approx at the mesh point x_i, we have $u_i = f(x_i)$. The collocation/interpolation using finite element basis functions gives exactly the same representa shown Section 4.4,

$$u(x_i) = c_i = f(x_i)\,.$$

How does a finite element Galerkin or least squares approximation differ from ıis straightforward interpolation of f? This is the question to be addressed ext. We now limit the scope to P1 elements since this is the element type that ives formulas closest to those arising in the finite difference method.

.2 Finite difference interpretation of a finite element approximation

he linear system arising from a Galerkin or least squares approximation reads ı general

$$\sum_{j\in\mathcal{I}_s} c_j(\psi_i,\psi_j) = (f,\psi_i), \quad i\in\mathcal{I}_s\,.$$

ı the finite element approximation we choose $\psi_i = \varphi_i$. With φ_i corresponding ɔ P1 elements and a uniform mesh of element length h we have in Section 3.6 ılculated the matrix with entries (φ_i,φ_j). Equation number i reads

$$\frac{h}{6}(u_{i-1} + 4u_i + u_{i+1}) = (f,\varphi_i)\,. \tag{84}$$

he first and last equation, corresponding to $i = 0$ and $i = N$ are slightly ifferent, see Section 4.6.

The finite difference counterpart to (84) is just $u_i = f_i$ as explained in ection 5.1. To easier compare this result to the finite element approach to pproximating functions, we can rewrite the left-hand side of (84) as

$$h(u_i + \frac{1}{6}(u_{i-1} - 2u_i + u_{i+1}))\,. \tag{85}$$

hinking in terms of finite differences, we can write this expression using finite ifference operator notation:

$$[h(u + \frac{h^2}{6}D_xD_xu)]_i,$$

hich is nothing but the standard discretization of

$$h(u + \frac{h^2}{6}u'')\,.$$

Before interpreting the approximation procedure as solving a differential quation, we need to work out what the right-hand side is in the context of P1 ements. Since φ_i is the linear function that is 1 at x_i and zero at all other odes, only the interval $[x_{i-1}, x_{i+1}]$ contribute to the integral on the right-hand de. This integral is naturally split into two parts according to (54):

$$(f,\varphi_i) = \int_{x_{i-1}}^{x_i} f(x)\frac{1}{h}(x - x_{i-1})dx + \int_{x_i}^{x_{i+1}} f(x)\frac{1}{h}(1 - (x - x_i))dx$$

However, if f is not known we cannot do much else with this expressio clear that many values of f around x_i contributes to the right-hand s just the single point value $f(x_i)$ as in the finite difference method.

To proceed with the right-hand side, we can turn to numerical inte schemes. The Trapezoidal method for (f,φ_i), based on sampling the in $f\varphi_i$ at the node points $x_i = ih$ gives

$$(f,\varphi_i) = \int_\Omega f\varphi_i dx \approx h\frac{1}{2}(f(x_0)\varphi_i(x_0) + f(x_N)\varphi_i(x_N)) + h\sum_{j=1}^{N-1} f(x_j)\varphi$$

Since φ_i is zero at all these points, except at x_i, the Trapezoidal rule c to one term:

$$(f,\varphi_i) \approx hf(x_i),$$

for $i = 1,\ldots,N-1$, which is the same result as with collocation/interp and of course the same result as in the finite difference method. For i $i = N$ we get contribution from only one element so

$$(f,\varphi_i) \approx \frac{1}{2}hf(x_i), \quad i = 0,\ i = N\,.$$

Simpson's rule with sample points also in the middle of the elem $x_{i+\frac{1}{2}} = (x_i + x_{i+1})/2$, can be written as

$$\int_\Omega g(x)dx \approx \frac{\tilde{h}}{3}\left(g(x_0) + 2\sum_{j=1}^{N-1} g(x_j) + 4\sum_{j=0}^{N-1} g(x_{j+\frac{1}{2}}) + f(x_{2N})\right)$$

where $\tilde{h} = h/2$ is the spacing between the sample points. Our integran $f\varphi_i$. For all the node points, $\varphi_i(x_j) = \delta_{ij}$, and therefore $\sum_{j=1}^{N-1} f(x_j)\varphi$ $f(x_i)$. At the midpoints, $\varphi_i(x_{i\pm\frac{1}{2}}) = 1/2$ and $\varphi_i(x_{j+\frac{1}{2}}) = 0$ for $j >$ $j < i-1$. Consequently,

$$\sum_{j=0}^{N-1} f(x_{j+\frac{1}{2}})\varphi_i(x_{j+\frac{1}{2}}) = \frac{1}{2}(fx_{j-\frac{1}{2}} + x_{j+\frac{1}{2}})\,.$$

When $1 \le i \le N-1$ we then get

$$(f,\varphi_i) \approx \frac{h}{3}(f_{i-\frac{1}{2}} + f_i + f_{i+\frac{1}{2}})\,.$$

his result shows that, with Simpson's rule, the finite element method operates ith the average of f over three points, while the finite difference method just pplies f at one point. We may interpret this as a "smearing" or smoothing of f y the finite element method.

We can now summarize our findings. With the approximation of (f, φ_i) by ıe Trapezoidal rule, P1 elements give rise to equations that can be expressed s a finite difference discretization of

$$u + \frac{h^2}{6}u'' = f, \quad u'(0) = u'(L) = 0, \tag{89}$$

xpressed with operator notation as

$$[u + \frac{h^2}{6}D_xD_xu = f]_i\,. \tag{90}$$

s $h \to 0$, the extra term proportional to u'' goes to zero, and the two methods re then equal.

With the Simpson's rule, we may say that we solve

$$[u + \frac{h^2}{6}D_xD_xu = \bar{f}]_i, \tag{91}$$

here $\bar{f}_i$ means the average $\frac{1}{3}(f_{i-1/2} + f_i + f_{i+1/2})$.

The extra term $\frac{h^2}{6}u''$ represents a smoothing effect: with just this term, e would find u by integrating f twice and thereby smooth f considerably. ı addition, the finite element representation of f involves an average, or a noothing, of f on the right-hand side of the equation system. If f is a noisy unction, direct interpolation $u_i = f_i$ may result in a noisy u too, but with a alerkin or least squares formulation and P1 elements, we should expect that u smoother than f unless h is very small.

The interpretation that finite elements tend to smooth the solution is valid ı applications far beyond approximation of 1D functions.

.3 Making finite elements behave as finite differences

/ith a simple trick, using numerical integration, we can easily produce the result $_i = f_i$ with the Galerkin or least square formulation with P1 elements. This is seful in many occasions when we deal with more difficult differential equations nd want the finite element method to have properties like the finite difference ıethod (solving standard linear wave equations is one primary example).

:omputations in physical space. We have already seen that applying the rapezoidal rule to the right-hand side (f, φ_i) simply gives f sampled at x_i. 'sing the Trapezoidal rule on the matrix entries $A_{i,j} = (\varphi_i, \varphi_j)$ involves a sum

$$\sum_k \varphi_i(x_k)\varphi_j(x_k),$$

but $\varphi_i(x_k) = \delta_{ik}$ and $\varphi_j(x_k) = \delta_{jk}$. The product $\varphi_i\varphi_j$ is then differe zero only when sampled at x_i and $i = j$. The Trapezoidal approximatio integral is then

$$(\varphi_i, \varphi_j) \approx h, \quad i = j,$$

and zero if $i \neq j$. This means that we have obtained a diagonal matrix! T and last diagonal elements, (φ_0, φ_0) and (φ_N, φ_N) get contribution on the first and last element, respectively, resulting in the approximate value $h/2$. The corresponding right-hand side also has a factor $1/2$ for i $i = N$. Therefore, the least squares or Galerkin approach with P1 eleme Trapezoidal integration results in

$$c_i = f_i, \quad i \in \mathcal{I}_s\,.$$

Simpsons's rule can be used to achieve a similar result for P2 elemen diagonal coefficient matrix, but with the previously derived average of f right-hand side.

Elementwise computations. Identical results to those above will we perform elementwise computations. The idea is to use the Trapezoi on the reference element for computing the element matrix and vector assembled, the same equations $c_i = f(x_i)$ arise. Exercise 19 encourages carry out the details.

Terminology. The matrix with entries (φ_i, φ_j) typically arises fron proportional to u in a differential equation where u is the unknown f This matrix is often called the *mass matrix*, because in the early day finite element method, the matrix arose from the mass times acceleratı in Newton's second law of motion. Making the mass matrix diagonal numerical integration, as demonstrated above, is a widely used techniqu called *mass lumping*. In time-dependent problems it can sometimes enha numerical accuracy and computational efficiency of the finite element However, there are also examples where mass lumping destroys accurac

6 A generalized element concept

So far, finite element computing has employed the `nodes` and `eleme` together with the definition of the basis functions in the reference Suppose we want to introduce a piecewise constant approximation with o function $\tilde{\varphi}_0(x) = 1$ in the reference element, corresponding to a $\varphi_i(x)$ f that is 1 on element number i and zero on all other elements. Although v associate the function value with a node in the middle of the elements, t no nodes at the ends, and the previous code snippets will not work bec cannot find the element boundaries from the `nodes` list.

.1 Cells, vertices, and degrees of freedom

/e now introduce *cells* as the subdomains $\Omega^{(e)}$ previously referred as elements. he cell boundaries are denoted as *vertices*. The reason for this name is that ells are recognized by their vertices in 2D and 3D. We also define a set of *degrees f freedom*, which are the quantities we aim to compute. The most common type f degree of freedom is the value of the unknown function u at some point. (For xample, we can introduce nodes as before and say the degrees of freedom are the alues of u at the nodes.) The basis functions are constructed so that they equal nity for one particular degree of freedom and zero for the rest. This property nsures that when we evaluate $u = \sum_j c_j\varphi_j$ for degree of freedom number i, we et $u = c_i$. Integrals are performed over cells, usually by mapping the cell of nterest to a *reference cell*.

With the concepts of cells, vertices, and degrees of freedom we increase the ecoupling of the geometry (cell, vertices) from the space of basis functions. /e will associate different sets of basis functions with a cell. In 1D, all cells re intervals, while in 2D we can have cells that are triangles with straight des, or any polygon, or in fact any two-dimensional geometry. Triangles and uadrilaterals are most common, though. The popular cell types in 3D are etrahedra and hexahedra.

.2 Extended finite element concept

he concept of a *finite element* is now

- a *reference cell* in a local reference coordinate system;
- a set of *basis functions* $\tilde{\varphi}_i$ defined on the cell;
- a set of *degrees of freedom* that uniquely determines the basis functions such that $\tilde{\varphi}_i = 1$ for degree of freedom number i and $\tilde{\varphi}_i = 0$ for all other degrees of freedom;
- a mapping between local and global degree of freedom numbers, here called the *dof map*;
- a geometric *mapping* of the reference cell onto to cell in the physical domain.

here must be a geometric description of a cell. This is trivial in 1D since the ell is an interval and is described by the interval limits, here called vertices. If ne cell is $\Omega^{(e)} = [x_L, x_R]$, vertex 0 is x_L and vertex 1 is x_R. The reference cell n 1D is $[-1, 1]$ in the reference coordinate system X.

The expansion of u over one cell is often used:

$$u(x) = \tilde{u}(X) = \sum_r c_r\tilde{\varphi}_r(X), \quad x \in \Omega^{(e)},\ X \in [-1, 1], \tag{92}$$

where the sum is taken over the numbers of the degrees of freedom and c value of u for degree of freedom number r.

Our previous P1, P2, etc., elements are defined by introducing $d + 1$ spaced nodes in the reference cell and saying that the degrees of freedom $d + 1$ function values at these nodes. The basis functions must be 1 at o and 0 at the others, and the Lagrange polynomials have exactly this p The nodes can be numbered from left to right with associated degrees of that are numbered in the same way. The degree of freedom mapping b what was previously represented by the `elements` lists. The cell mappir same affine mapping (61) as before.

6.3 Implementation

Implementationwise,

- we replace `nodes` by `vertices`;
- we introduce `cells` such that `cell[e][r]` gives the mapping fro vertex `r` in cell `e` to the global vertex number in `vertices`;
- we replace `elements` by `dof_map` (the contents are the same for ments).

Consider the example from Section 3.1 where $\Omega = [0, 1]$ is divided into t $\Omega^{(0)} = [0, 0.4]$ and $\Omega^{(1)} = [0.4, 1]$, as depicted in Figure 16. The vert $[0, 0.4, 1]$. Local vertex 0 and 1 are 0 and 0.4 in cell 0 and 0.4 and 1 in c P2 element means that the degrees of freedom are the value of u at three spaced points (nodes) in each cell. The data structures become

```
vertices = [0, 0.4, 1]
cells = [[0, 1], [1, 2]]
dof_map = [[0, 1, 2], [2, 3, 4]]
```

If we would approximate f by piecewise constants, known as P0 elem simply introduce one point or node in an element, preferably $X = 0$, an one degree of freedom, which is the function value at this node. Morec set $\tilde{\varphi}_0(X) = 1$. The `cells` and `vertices` arrays remain the same, but c is altered:

```
dof_map = [[0], [1]]
```

We use the `cells` and `vertices` lists to retrieve information on the g of a cell, while `dof_map` is the $q(e, r)$ mapping introduced earlier in the a of element matrices and vectors. For example, the `Omega_e` variable (repr the cell interval) in previous code snippets must now be computed as

```
Omega_e = [vertices[cells[e][0], vertices[cells[e][1]]
```

he assembly is done by

```
A[dof_map[e][r], dof_map[e][s]] += A_e[r,s]
b[dof_map[e][r]] += b_e[r]
```

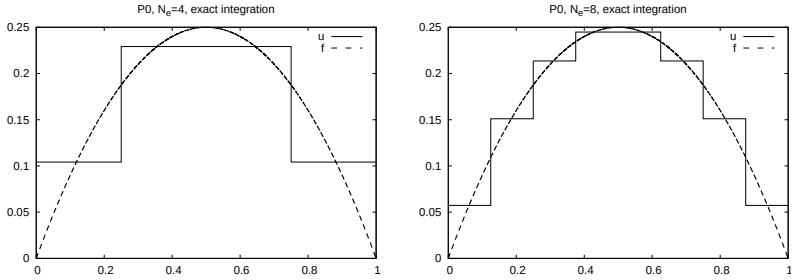
We will hereafter drop the `nodes` and `elements` arrays and work exculsively ith `cells`, `vertices`, and `dof_map`. The module `fe_approx1D_numint.py` ow replaces the module `fe_approx1D` and offers similar functions that work ith the new concepts:

```
from fe_approx1D_numint import *
x = sp.Symbol('x')
f = x*(1 - x)
N_e = 10
vertices, cells, dof_map = mesh_uniform(N_e, d=3, Omega=[0,1])
phi = [basis(len(dof_map[e])-1) for e in range(N_e)]
A, b = assemble(vertices, cells, dof_map, phi, f)
c = np.linalg.solve(A, b)
# Make very fine mesh and sample u(x) on this mesh for plotting
x_u, u = u_glob(c, vertices, cells, dof_map,
                resolution_per_element=51)
plot(x_u, u)
```

hese steps are offered in the `approximate` function, which we here apply to see ow well four P0 elements (piecewise constants) can approximate a parabola:

```
from fe_approx1D_numint import *
x=sp.Symbol("x")
for N_e in 4, 8:
    approximate(x*(1-x), d=0, N_e=N_e, Omega=[0,1])
```

igure 32 shows the result.



igure 32: Approximation of a parabola by 4 (left) and 8 (right) P0 elements.

6.4 Computing the error of the approximation

So far we have focused on computing the coefficients c_j in the approx $u(x) = \sum_j c_j \varphi_j$ as well as on plotting u and f for visual comparison. quantitative comparison needs to investigate the error $e(x) = f(x) - u$ mostly want a single number to reflect the error and use a norm for this usually the L^2 norm

$$||e||_{L^2} = \left(\int_\Omega e^2 dx \right)^{1/2}.$$

Since the finite element approximation is defined for all $x \in \Omega$, and interested in how $u(x)$ deviates from $f(x)$ through all the elements, we ca integrate analytically or use an accurate numerical approximation. The more convenient as it is a generally feasible and simple approach. The ic sample $e(x)$ at a large number of points in each element. The function in the `fe_approx1D_numint` module does this for $u(x)$ and returns an with coordinates and an array `u` with the u values:

```
x, u = u_glob(c, vertices, cells, dof_map,
              resolution_per_element=101)
e = f(x) - u
```

Let us use the Trapezoidal method to approximate the integral. Because elements may have different lengths, the `x` array has a non-uniformly dis set of coordinates. Also, the `u_glob` function works in an element by fashion such that coordinates at the boundaries between elements appe We therefore need to use a "raw" version of the Trapezoidal rule where add up all the trapezoids:

$$\int_\Omega g(x)dx \approx \sum_{j=0}^{n-1} \frac{1}{2}(g(x_j) + g(x_{j+1}))(x_{j+1} - x_j),$$

if $x_0, \ldots, x_n$ are all the coordinates in `x`. In vectorized Python code,

```
g_x = g(x)
integral = 0.5*np.sum((g_x[:-1] + g_x[1:])*(x[1:] - x[:-1]))
```

Computing the L^2 norm of the error, here named `E`, is now achieved by

```
e2 = e**2
E = np.sqrt(0.5*np.sum((e2[:-1] + e2[1:])*(x[1:] - x[:-1])))
```

How does the error depend on h and d?

Theory and experiments show that the least squares or projection/Gal method in combination with Pd elements of equal length h has an e

$$||e||_{L^2} = Ch^{d+1}, \tag{93}$$

where C is a constant depending on f, but not on h or d.

.5 Example: Cubic Hermite polynomials

he finite elements considered so far represent u as piecewise polynomials with iscontinuous derivatives at the cell boundaries. Sometimes it is desirable to ave continuous derivatives. A primary examples is the solution of differential quations with fourth-order derivatives where standard finite element formula- ons lead to a need for basis functions with continuous first-order derivatives. he most common type of such basis functions in 1D is the so-called cubic ermite polynomials. The construction of such polynomials, as explained next, ill further exemplify the concepts of a cell, vertex, degree of freedom, and dof ıap.

Given a reference cell $[-1, 1]$, we seek cubic polynomials with the values of ıe *function* and its *first-order derivative* at $X = -1$ and $X = 1$ as the four egrees of freedom. Let us number the degrees of freedom as

- 0: value of function at $X = -1$
- 1: value of first derivative at $X = -1$
- 2: value of function at $X = 1$
- 3: value of first derivative at $X = 1$

y having the derivatives as unknowns, we ensure that the derivative of a basis unction in two neighboring elements is the same at the node points.

The four basis functions can be written in a general form

$$\tilde{\varphi}_i(X) = \sum_{j=0}^{3} C_{i,j} X^j,$$

ith four coefficients $C_{i,j}$, $j = 0, 1, 2, 3$, to be determined for each i. The onstraints that basis function number i must be 1 for degree of freedom number and zero for the other three degrees of freedom, gives four equations to determine $_{i,j}$ for each i. In mathematical detail,

$$\begin{aligned}
\tilde{\varphi}_0(-1) = 1, &\quad \tilde{\varphi}_0(1) = \tilde{\varphi}_0'(-1) = \tilde{\varphi}_i'(1) = 0,\\
\tilde{\varphi}_1'(-1) = 1, &\quad \tilde{\varphi}_1(-1) = \tilde{\varphi}_1(1) = \tilde{\varphi}_1'(1) = 0,\\
\tilde{\varphi}_2(1) = 1, &\quad \tilde{\varphi}_2(-1) = \tilde{\varphi}_2'(-1) = \tilde{\varphi}_2'(1) = 0,\\
\tilde{\varphi}_3'(1) = 1, &\quad \tilde{\varphi}_3(-1) = \tilde{\varphi}_3'(-1) = \tilde{\varphi}_3(1) = 0\,.
\end{aligned}$$

hese four 4×4 linear equations can be solved, yielding the following formulas ɔr the cubic basis functions:

$$\begin{aligned}
\tilde{\varphi}_0(X) &= 1 - \frac{3}{4}(X+1)^2 + \frac{1}{4}(X+1)^3\\
\tilde{\varphi}_1(X) &= -(X+1)(1 - \frac{1}{2}(X+1))^2\\
\tilde{\varphi}_2(X) &= \frac{3}{4}(X+1)^2 - \frac{1}{2}(X+1)^3\\
\tilde{\varphi}_3(X) &= -\frac{1}{2}(X+1)(\frac{1}{2}(X+1)^2 - (X+1))
\end{aligned}$$

The construction of the dof map needs a scheme for numbering th degrees of freedom. A natural left-to-right numbering has the function vertex x_i as degree of freedom number $2i$ and the value of the derivati as degree of freedom number $2i + 1$, $i = 0, \ldots, N_e + 1$.

7 Numerical integration

Finite element codes usually apply numerical approximations to integral the integrands in the coefficient matrix often are (lower-order) polyı integration rules that can integrate polynomials exactly are popular.

The numerical integration rules can be expressed in a common form

$$\int_{-1}^{1} g(X)dX \approx \sum_{j=0}^{M} w_j g(\bar{X}_j),$$

where $\bar{X}_j$ are *integration points* and w_j are *integration weights*, $j = 0$ Different rules correspond to different choices of points and weights.

The very simplest method is the *Midpoint rule*,

$$\int_{-1}^{1} g(X)dX \approx 2g(0), \quad \bar{X}_0 = 0,\ w_0 = 2,$$

which integrates linear functions exactly.

7.1 Newton-Cotes rules

The Newton-Cotes[8] rules are based on a fixed uniform distribution integration points. The first two formulas in this family are the well *Trapezoidal rule*,

$$\int_{-1}^{1} g(X)dX \approx g(-1) + g(1), \quad \bar{X}_0 = -1,\ \bar{X}_1 = 1,\ w_0 = w_1 = 1,$$

[8] http://en.wikipedia.org/wiki/Newton%E2%80%93Cotes_formulas

nd *Simpson's rule*,

$$\int_{-1}^{1} g(X)dX \approx \frac{1}{3}\left(g(-1) + 4g(0) + g(1)\right), \tag{102}$$

here

$$\bar{X}_0 = -1,\ \bar{X}_1 = 0,\ \bar{X}_2 = 1,\ w_0 = w_2 = \frac{1}{3},\ w_1 = \frac{4}{3}. \tag{103}$$

ewton-Cotes rules up to five points is supported in the module file `numint.py`[9].

For higher accuracy one can divide the reference cell into a set of subintervals nd use the rules above on each subinterval. This approach results in *composite* ules, well-known from basic introductions to numerical integration of $\int_a^b f(x)dx$.

.2 Gauss-Legendre rules with optimized points

lore accurate rules, for a given M, arise if the location of the integration points re optimized for polynomial integrands. The Gauss-Legendre rules[10] (also nown as Gauss-Legendre quadrature or Gaussian quadrature) constitute one ich class of integration methods. Two widely applied Gauss-Legendre rules in is family have the choice

$$M = 1: \quad \bar{X}_0 = -\frac{1}{\sqrt{3}},\ \bar{X}_1 = \frac{1}{\sqrt{3}},\ w_0 = w_1 = 1 \tag{104}$$

$$M = 2: \quad \bar{X}_0 = -\sqrt{\frac{3}{5}},\ \bar{X}_0 = 0,\ \bar{X}_2 = \sqrt{\frac{3}{5}},\ w_0 = w_2 = \frac{5}{9},\ w_1 = \frac{8}{9}. \tag{105}$$

hese rules integrate 3rd and 5th degree polynomials exactly. In general, an I-point Gauss-Legendre rule integrates a polynomial of degree $2M+1$ exactly. he code `numint.py` contains a large collection of Gauss-Legendre rules.

5 Approximation of functions in 2D

ll the concepts and algorithms developed for approximation of 1D functions (x) can readily be extended to 2D functions $f(x,y)$ and 3D functions $f(x,y,z)$. asically, the extensions consists of defining basis functions $\psi_i(x,y)$ or $\psi_i(x,y,z)$ ver some domain Ω, and for the least squares and Galerkin methods, the ntegration is done over Ω.

As in 1D, the least squares and projection/Galerkin methods two lead to near systems

[9] `http://tinyurl.com/nm5587k/fem/numint.py`
[10] `http://en.wikipedia.org/wiki/Gaussian_quadrature`

$$\sum_{j\in\mathcal{I}_s} A_{i,j}c_j = b_i, \quad i \in \mathcal{I}_s,$$

$$A_{i,j} = (\psi_i, \psi_j),$$
$$b_i = (f, \psi_i),$$

where the inner product of two functions $f(x,y)$ and $g(x,y)$ is defined co analogously to the 1D case (24):

$$(f,g) = \int_\Omega f(x,y)g(x,y)dxdy$$

8.1 2D basis functions as tensor products of 1D fun

One straightforward way to construct a basis in 2D is to combine 1 functions. Say we have the 1D vector space

$$V_x = \text{span}\{\hat{\psi}_0(x), \ldots, \hat{\psi}_{N_x}(x)\}.$$

A similar space for variation in y can be defined,

$$V_y = \text{span}\{\hat{\psi}_0(y), \ldots, \hat{\psi}_{N_y}(y)\}.$$

We can then form 2D basis functions as *tensor products* of 1D basis fu

Tensor products.

Given two vectors $a = (a_0, \ldots, a_M)$ and $b = (b_0, \ldots, b_N)$, their *outer t product*, also called the *dyadic product*, is $p = a \otimes b$, defined through

$$p_{i,j} = a_i b_j, \quad i = 0, \ldots, M,\ j = 0, \ldots, N.$$

In the tensor terminology, a and b are first-order tensors (vectors wit index, also termed rank-1 tensors), and then their outer tensor produ a second-order tensor (matrix with two indices, also termed rank-2 te The corresponding *inner tensor product* is the well-known scalar o product of two vectors: $p = a \cdot b = \sum_{j=0}^{N} a_j b_j$. Now, p is a rank-0 te

Tensors are typically represented by arrays in computer code. I above example, a and b are represented by one-dimensional arra length M and N, respectively, while $p = a \otimes b$ must be represented two-dimensional array of size $M \times N$.

Tensor products[a] can be used in a variety of context.

[a] `http://en.wikipedia.org/wiki/Tensor_product`

Given the vector spaces V_x and V_y as defined in (107) and (108), th product space $V = V_x \otimes V_y$ has a basis formed as the tensor produc

asis for V_x and V_y. That is, if $\{\varphi_i(x)\}_{i\in\mathcal{I}_x}$ and $\{\varphi_i(y)\}_{i\in\mathcal{I}_y}$ are basis for V_x nd V_y, respectively, the elements in the basis for V arise from the tensor roduct: $\{\varphi_i(x)\varphi_j(y)\}_{i\in\mathcal{I}_x,j\in\mathcal{I}_y}$. The index sets are $I_x = \{0,\ldots,N_x\}$ and $_i = \{0,\ldots,N_y\}$.

The notation for a basis function in 2D can employ a double index as in

$$\psi_{p,q}(x,y) = \hat{\psi}_p(x)\hat{\psi}_q(y), \quad p\in\mathcal{I}_x, q\in\mathcal{I}_y\,.$$

he expansion for u is then written as a double sum

$$u = \sum_{p\in\mathcal{I}_x}\sum_{q\in\mathcal{I}_y} c_{p,q}\psi_{p,q}(x,y)\,.$$

lternatively, we may employ a single index,

$$\psi_i(x,y) = \hat{\psi}_p(x)\hat{\psi}_q(y),$$

nd use the standard form for u,

$$u = \sum_{j\in\mathcal{I}_s} c_j\psi_j(x,y)\,.$$

he single index is related to the double index through $i = pN_y+q$ or $i = qN_x+p$.

.2 Example: Polynomial basis in 2D

uppose we choose $\hat{\psi}_p(x) = x^p$, and try an approximation with $N_x = N_y = 1$:

$$\psi_{0,0} = 1, \quad \psi_{1,0} = x, \quad \psi_{0,1} = y, \quad \psi_{1,1} = xy\,.$$

sing a mapping to one index like $i = qN_x + p$, we get

$$\psi_0 = 1, \quad \psi_1 = x, \quad \psi_2 = y, \quad \psi_3 = xy\,.$$

With the specific choice $f(x,y) = (1+x^2)(1+2y^2)$ on $\Omega = [0,L_x]\times[0,L_y]$, e can perform actual calculations:

$$\begin{aligned}
A_{0,0} &= (\psi_0,\psi_0) = \int_0^{L_y}\int_0^{L_x}\psi_0(x,y)^2dxdy = \int_0^{L_y}\int_0^{L_x}dxdy = L_xL_y,\\
A_{1,0} &= (\psi_1,\psi_0) = \int_0^{L_y}\int_0^{L_x}xdxdy = \frac{1}{2}L_x^2L_y,\\
A_{0,1} &= (\psi_0,\psi_1) = \int_0^{L_y}\int_0^{L_x}ydxdy = \frac{1}{2}L_y^2L_x,\\
A_{0,1} &= (\psi_0,\psi_1) = \int_0^{L_y}\int_0^{L_x}xydxdy = \int_0^{L_y}ydy\int_0^{L_x}xdx = \frac{1}{4}L_y^2L_x^2\,.
\end{aligned}$$

he right-hand side vector has the entries

$$\begin{aligned}
b_0 &= (\psi_0,f) = \int_0^{L_y}\int_0^{L_x}1\cdot(1+x^2)(1+2y^2)dxdy\\
&= \int_0^{L_y}(1+2y^2)dy\int_0^{L_x}(1+x^2)dx = (L_y+\frac{2}{3}L_y^3)(L_x+\frac{1}{3}L_x^3)\\
b_1 &= (\psi_1,f) = \int_0^{L_y}\int_0^{L_x}x(1+x^2)(1+2y^2)dxdy\\
&= \int_0^{L_y}(1+2y^2)dy\int_0^{L_x}x(1+x^2)dx = (L_y+\frac{2}{3}L_y^3)(\frac{1}{2}L_x^2+\frac{1}{4}L_x^4)\\
b_2 &= (\psi_2,f) = \int_0^{L_y}\int_0^{L_x}y(1+x^2)(1+2y^2)dxdy\\
&= \int_0^{L_y}y(1+2y^2)dy\int_0^{L_x}(1+x^2)dx = (\frac{1}{2}L_y+\frac{1}{2}L_y^4)(L_x+\frac{1}{3}L_x^3)\\
b_3 &= (\psi_2,f) = \int_0^{L_y}\int_0^{L_x}xy(1+x^2)(1+2y^2)dxdy\\
&= \int_0^{L_y}y(1+2y^2)dy\int_0^{L_x}x(1+x^2)dx = (\frac{1}{2}L_y^2+\frac{1}{2}L_y^4)(\frac{1}{2}L_x^2+\frac{1}{4}I
\end{aligned}$$

There is a general pattern in these calculations that we can explo arbitrary matrix entry has the formula

$$\begin{aligned}
A_{i,j} &= (\psi_i,\psi_j) = \int_0^{L_y}\int_0^{L_x}\psi_i\psi_jdxdy\\
&= \int_0^{L_y}\int_0^{L_x}\psi_{p,q}\psi_{r,s}dxdy = \int_0^{L_y}\int_0^{L_x}\hat{\psi}_p(x)\hat{\psi}_q(y)\hat{\psi}_r(x)\hat{\psi}_s(y)dx\\
&= \int_0^{L_y}\hat{\psi}_q(y)\hat{\psi}_s(y)dy\int_0^{L_x}\hat{\psi}_p(x)\hat{\psi}_r(x)dx\\
&= \hat{A}^{(x)}_{p,r}\hat{A}^{(y)}_{q,s},
\end{aligned}$$

where

$$\hat{A}^{(x)}_{p,r} = \int_0^{L_x}\hat{\psi}_p(x)\hat{\psi}_r(x)dx, \quad \hat{A}^{(y)}_{q,s} = \int_0^{L_y}\hat{\psi}_q(y)\hat{\psi}_s(y)dy,$$

are matrix entries for one-dimensional approximations. Moreover, $i =$ and $j = sN_y + r$.

With $\hat{\psi}_p(x) = x^p$ we have

$$\hat{A}^{(x)}_{p,r} = \frac{1}{p+r+1}L_x^{p+r+1}, \quad \hat{A}^{(y)}_{q,s} = \frac{1}{q+s+1}L_y^{q+s+1},$$

and

$$A_{i,j} = \hat{A}^{(x)}_{p,r}\hat{A}^{(y)}_{q,s} = \frac{1}{p+r+1}L_x^{p+r+1}\frac{1}{q+s+1}L_y^{q+s+1},$$

or $p, r \in \mathcal{I}_x$ and $q, s \in \mathcal{I}_y$.

Corresponding reasoning for the right-hand side leads to

$$\begin{aligned}
b_i = (\psi_i, f) &= \int_0^{L_y}\int_0^{L_x} \psi_i f\, dxdx \\
&= \int_0^{L_y}\int_0^{L_x} \hat{\psi}_p(x)\hat{\psi}_q(y) f\, dxdx \\
&= \int_0^{L_y} \hat{\psi}_q(y)(1+2y^2)dy \int_0^{L_y} \hat{\psi}_p(x)x^p(1+x^2)dx \\
&= \int_0^{L_y} y^q(1+2y^2)dy \int_0^{L_y} x^p(1+x^2)dx \\
&= (\frac{1}{q+1}L_y^{q+1} + \frac{2}{q+3}L_y^{q+3})(\frac{1}{p+1}L_x^{p+1} + \frac{2}{q+3}L_x^{p+3})
\end{aligned}$$

Choosing $L_x = L_y = 2$, we have

$$A = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 4 & \frac{16}{3} & 4 & \frac{16}{3} \\ 4 & 4 & \frac{16}{3} & \frac{16}{3} \\ 4 & \frac{16}{3} & \frac{16}{3} & \frac{64}{9} \end{bmatrix}, \quad b = \begin{bmatrix} \frac{308}{9} \\ \frac{140}{3} \\ 44 \\ 60 \end{bmatrix}, \quad c = \begin{bmatrix} -\frac{1}{9} \\ \frac{4}{3} \\ -\frac{2}{3} \\ 8 \end{bmatrix}.$$

Figure 33 illustrates the result.

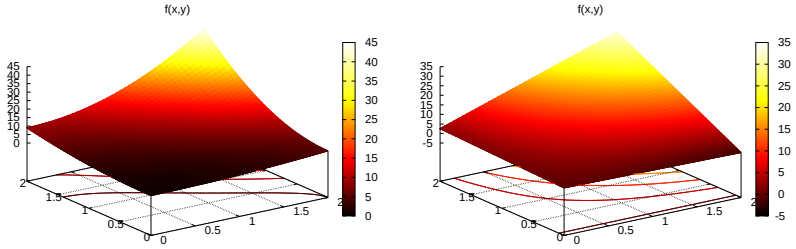


Figure 33: Approximation of a 2D quadratic function (left) by a 2D bilinear function (right) using the Galerkin or least squares method.

.3 Implementation

The `least_squares` function from Section 2.8 and/or the file `approx1D.py`[11] can with very small modifications solve 2D approximation problems. First,

[11] `http://tinyurl.com/nm5587k/fem/fe_approx1D.py`

let `Omega` now be a list of the intervals in x and y direction. For example, $\Omega = [0, L_x] \times [0, L_y]$ can be represented by `Omega = [[0, L_x], [0, L_y]]`.

Second, the symbolic integration must be extended to 2D:

```
import sympy as sp

integrand = psi[i]*psi[j]
I = sp.integrate(integrand,
                 (x, Omega[0][0], Omega[0][1]),
                 (y, Omega[1][0], Omega[1][1]))
```

provided `integrand` is an expression involving the `sympy` symbols `x` and `y`. The 2D version of numerical integration becomes

```
if isinstance(I, sp.Integral):
    integrand = sp.lambdify([x,y], integrand)
    I = sp.mpmath.quad(integrand,
                       [Omega[0][0], Omega[0][1]],
                       [Omega[1][0], Omega[1][1]])
```

The right-hand side integrals are modified in a similar way.

Third, we must construct a list of 2D basis functions. Here are two examples based on tensor products of 1D "Taylor-style" polynomials x^i and sine functions $\sin((i+1)\pi x)$:

```
def taylor(x, y, Nx, Ny):
    return [x**i*y**j for i in range(Nx+1) for j in range(Ny+1)]

def sines(x, y, Nx, Ny):
    return [sp.sin(sp.pi*(i+1)*x)*sp.sin(sp.pi*(j+1)*y)
            for i in range(Nx+1) for j in range(Ny+1)]
```

The complete code appears in `approx2D.py`[12].

The previous hand calculation where a quadratic f was approximated by a bilinear function can be computed symbolically by

```
>>> from approx2D import *
>>> f = (1+x**2)*(1+2*y**2)
>>> psi = taylor(x, y, 1, 1)
>>> Omega = [[0, 2], [0, 2]]
>>> u, c = least_squares(f, psi, Omega)
>>> print u
8*x*y - 2*x/3 + 4*y/3 - 1/9
>>> print sp.expand(f)
2*x**2*y**2 + x**2 + 2*y**2 + 1
```

We may continue with adding higher powers to the basis:

```
>>> psi = taylor(x, y, 2, 2)
>>> u, c = least_squares(f, psi, Omega)
>>> print u
2*x**2*y**2 + x**2 + 2*y**2 + 1
>>> print u-f
0
```

[12] `http://tinyurl.com/nm5587k/fem/fe_approx2D.py`

or $N_x \geq 2$ and $N_y \geq 2$ we recover the exact function f, as expected, since in iat case $f \in V$ (see Section 2.5).

.4 Extension to 3D

xtension to 3D is in principle straightforward once the 2D extension is unerstood. The only major difference is that we need the repeated outer tensor roduct,

$$V = V_x \otimes V_y \otimes V_z \,.$$

ı general, given vectors (first-order tensors) $a^{(q)} = (a_0^{(q)}, \ldots, a_{N_q}^{(q)}$, $q = 0, \ldots, m$, ıe tensor product $p = a^{(0)} \otimes \cdots \otimes a^m$ has elements

$$p_{i_0,i_1,\ldots,i_m} = a_{i_1}^{(0)} a_{i_1}^{(1)} \cdots a_{i_m}^{(m)} \,.$$

he basis functions in 3D are then

$$\psi_{p,q,r}(x,y,z) = \hat{\psi}_p(x)\hat{\psi}_q(y)\hat{\psi}_r(z),$$

ith $p \in \mathcal{I}_x$, $q \in \mathcal{I}_y$, $r \in \mathcal{I}_z$. The expansion of u becomes

$$u(x,y,z) = \sum_{p\in\mathcal{I}_x} \sum_{q\in\mathcal{I}_y} \sum_{r\in\mathcal{I}_z} c_{p,q,r}\psi_{p,q,r}(x,y,z) \,.$$

single index can be introduced also here, e.g., $i = N_x N_y r + q_N x + p$, $u = \sum_i c_i \psi_i(x,y,z)$.

Use of tensor product spaces.

Constructing a multi-dimensional space and basis from tensor products of 1D spaces is a standard technique when working with global basis functions. In the world of finite elements, constructing basis functions by tensor products is much used on quadrilateral and hexahedra cell shapes, but not on triangles and tetrahedra. Also, the global finite element basis functions are almost exclusively denoted by a single index and not by the natural tuple of indices that arises from tensor products.

Finite elements in 2D and 3D

inite element approximation is particularly powerful in 2D and 3D because ıe method can handle a geometrically complex domain Ω with ease. The rincipal idea is, as in 1D, to divide the domain into cells and use polynomials ɔr approximating a function over a cell. Two popular cell shapes are triangles nd the quadrilaterals. Figures 34, 35, and 36 provide examples. P1 elements

means linear functions $(a_0 + a_1 x + a_2 y)$ over triangles, while Q1 elemer bilinear functions $(a_0 + a_1 x + a_2 y + a_3 xy)$ over rectangular cells. High elements can easily be defined.

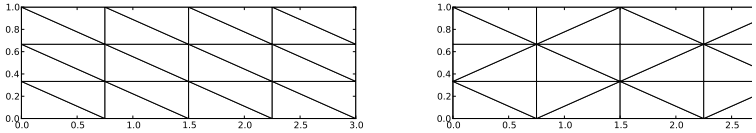


Figure 34: Examples on 2D P1 elements.

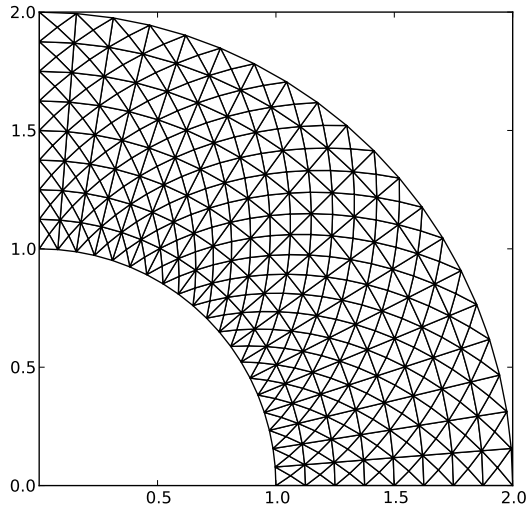


Figure 35: Examples on 2D P1 elements in a deformed geometry

9.1 Basis functions over triangles in the physical dc

Cells with triangular shape will be in main focus here. With the P1 tri element, u is a linear function over each cell, as depicted in Figure 3 discontinuous derivatives at the cell boundaries.

We give the vertices of the cells global and local numbers as in 1 degrees of freedom in the P1 element are the function values at a set o which are the three vertices. The basis function $\varphi_i(x,y)$ is then 1 at th with global vertex number i and zero at all other vertices. On an elem three degrees of freedom uniquely determine the linear basis functions

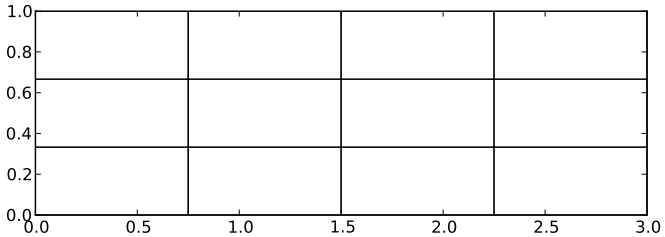


Figure 36: Examples on 2D Q1 elements.

ement, as usual. The global $\varphi_i(x,y)$ function is then a combination of the linear nctions (planar surfaces) over all the neighboring cells that have vertex number in common. Figure 38 tries to illustrate the shape of such a "pyramid"-like nction.

lement matrices and vectors. As in 1D, we split the integral over Ω into sum of integrals over cells. Also as in 1D, φ_i overlaps φ_j (i.e., $\varphi_i\varphi_j \neq 0$) if nd only if i and j are vertices in the same cell. Therefore, the integral of $\varphi_i\varphi_j$ ver an element is nonzero only when i and j run over the vertex numbers in ıe element. These nonzero contributions to the coefficient matrix are, as in 1D, ollected in an element matrix. The size of the element matrix becomes 3×3 nce there are three degrees of freedom that i and j run over. Again, as in 1D, e number the local vertices in a cell, starting at 0, and add the entries in the ement matrix into the global system matrix, exactly as in 1D. All details and ode appear below.

.2 Basis functions over triangles in the reference cell

s in 1D, we can define the basis functions and the degrees of freedom in a eference cell and then use a mapping from the reference coordinate system to ıe physical coordinate system. We also have a mapping of local degrees of eedom numbers to global degrees of freedom numbers.

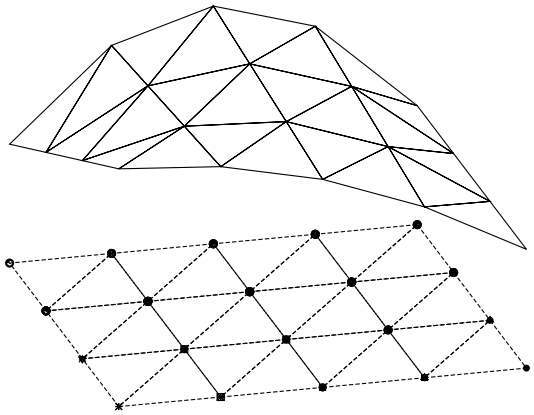


Figure 37: Example on piecewise linear 2D functions defined on trial

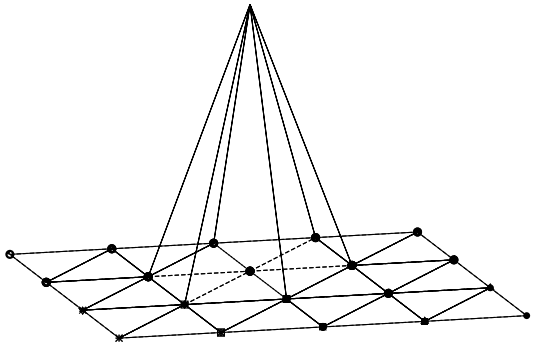
The reference cell in an (X,Y) coordinate system has vertices $(0,0$ and $(0,1)$, corresponding to local vertex numbers 0, 1, and 2, respective P1 element has linear functions $\tilde{\varphi}_r(X,Y)$ as basis functions, $r = 0,1,2$. linear function $\tilde{\varphi}_r(X,Y)$ in 2D is on the form $C_{r,0} + C_{r,1}X + C_{r,2}Y$, an has three parameters $C_{r,0}$, $C_{r,1}$, and $C_{r,2}$, we need three degrees of f These are in general taken as the function values at a set of nodes. For element the set of nodes is the three vertices. Figure 39 displays the g of the element and the location of the nodes.

Requiring $\tilde{\varphi}_r = 1$ at node number r and $\tilde{\varphi}_r = 0$ at the two other nod three linear equations to determine $C_{r,0}$, $C_{r,1}$, and $C_{r,2}$. The result is

$$\tilde{\varphi}_0(X,Y) = 1 - X - Y,$$
$$\tilde{\varphi}_1(X,Y) = X,$$
$$\tilde{\varphi}_2(X,Y) = Y$$

Higher-order approximations are obtained by increasing the polynomi adding additional nodes, and letting the degrees of freedom be functio at the nodes. Figure 40 shows the location of the six nodes in the P2 e

A polynomial of degree p in X and Y has $n_p = (p+1)(p+2)/2$ ter hence needs n_p nodes. The values at the nodes constitute n_p degrees of The location of the nodes for $\tilde{\varphi}_r$ up to degree 6 is displayed in Figure 4



igure 38: Example on a piecewise linear 2D basis function over a patch of riangles.

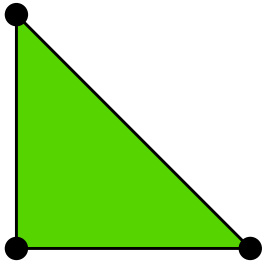


Figure 39: 2D P1 element.

The generalization to 3D is straightforward: the reference element is a etrahedron[13] with vertices $(0,0,0)$, $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$ in a X,Y,Z eference coordinate system. The P1 element has its degrees of freedom as four odes, which are the four vertices, see Figure 42. The P2 element adds additional odes along the edges of the cell, yielding a total of 10 nodes and degrees of eedom, see Figure 43.

The interval in 1D, the triangle in 2D, the tetrahedron in 3D, and its eneralizations to higher space dimensions are known as *simplex* cells (the eometry) or *simplex* elements (the geometry, basis functions, degrees of freedom,

[13] `http://en.wikipedia.org/wiki/Tetrahedron`

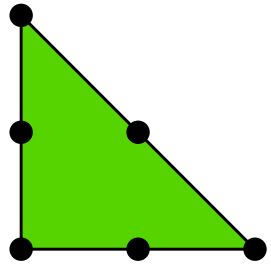


Figure 40: 2D P2 element.

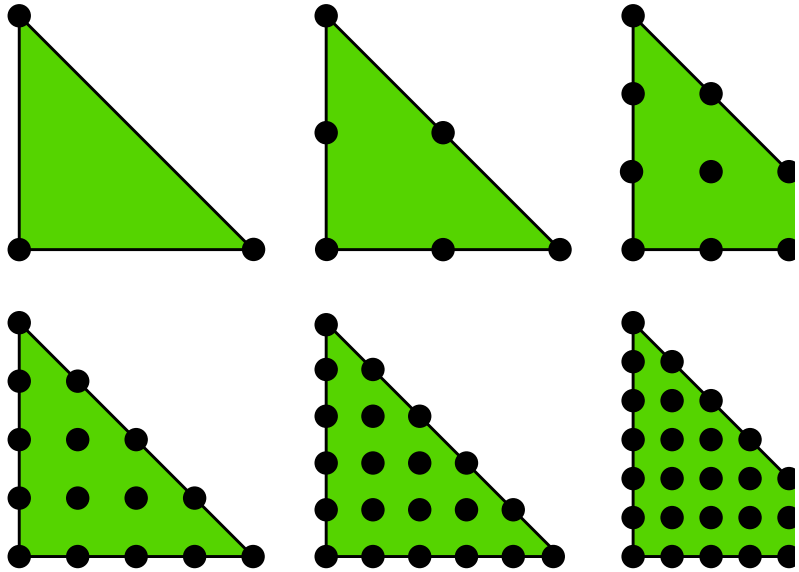


Figure 41: 2D P1, P2, P3, P4, P5, and P6 elements.

etc.). The plural forms simplices[14] and simplexes are also a much used terms when referring to this type of cells or elements. The side of a sir called a *face*, while the tetrahedron also has *edges*.

Acknowledgment. Figures 39 to 43 are created by Anders Logg an from the FEniCS book[15]: *Automated Solution of Differential Equation Finite Element Method*, edited by A. Logg, K.-A. Mardal, and G. N published by Springer[16], 2012.

[14] `http://en.wikipedia.org/wiki/Simplex`
[15] `https://launchpad.net/fenics-book`
[16] `http://goo.gl/lbyVMH`

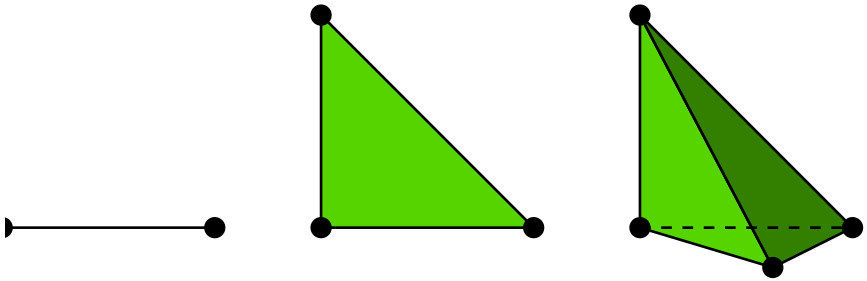


Figure 42: P1 elements in 1D, 2D, and 3D.

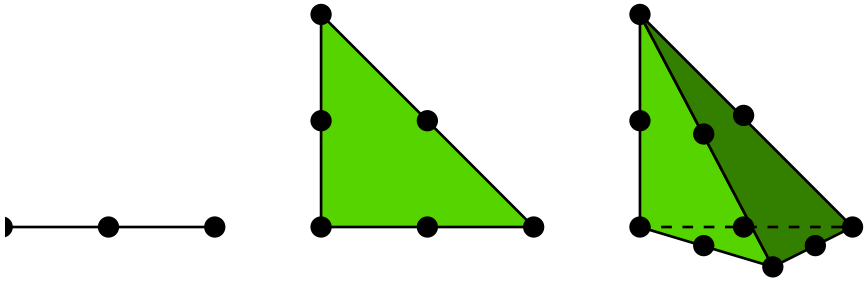


Figure 43: P2 elements in 1D, 2D, and 3D.

.3 Affine mapping of the reference cell

et $\tilde{\varphi}_r^{(1)}$ denote the basis functions associated with the P1 element in 1D, 2D, or D, and let $\boldsymbol{x}_{q(e,r)}$ be the physical coordinates of local vertex number r in cell e. urthermore, let $\boldsymbol{X}$ be a point in the reference coordinate system corresponding o the point $\boldsymbol{x}$ in the physical coordinate system. The affine mapping of any $\boldsymbol{X}$ nto $\boldsymbol{x}$ is then defined by

$$\boldsymbol{x} = \sum_r \tilde{\varphi}_r^{(1)}(\boldsymbol{X})\boldsymbol{x}_{q(e,r)}, \tag{112}$$

here r runs over the local vertex numbers in the cell. The affine mapping ssentially stretches, translates, and rotates the triangle. Straight or planar ces of the reference cell are therefore mapped onto straight or planar faces n the physical coordinate system. The mapping can be used for both P1 and igher-order elements, but note that the mapping itself always applies the P1 asis functions.

.4 Isoparametric mapping of the reference cell

nstead of using the P1 basis functions in the mapping (112), we may use the asis functions of the actual Pd element:

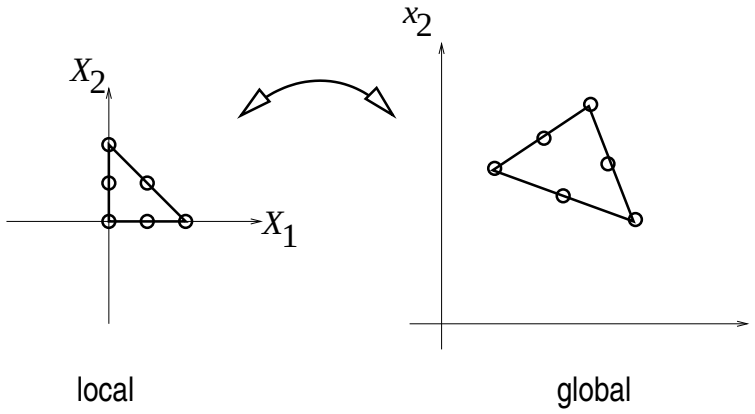


Figure 44: Affine mapping of a P1 element.

$$\boldsymbol{x} = \sum_r \tilde{\varphi}_r(\boldsymbol{X})\boldsymbol{x}_{q(e,r)},$$

where r runs over all nodes, i.e., all points associated with the degrees of This is called an *isoparametric mapping*. For P1 elements it is iden the affine mapping (112), but for higher-order elements the mapping straight or planar faces of the reference cell will result in a *curved* fac physical coordinate system. For example, when we use the basis functior triangular P2 element in 2D in (113), the straight faces of the reference are mapped onto curved faces of parabolic shape in the physical coo system, see Figure 45.

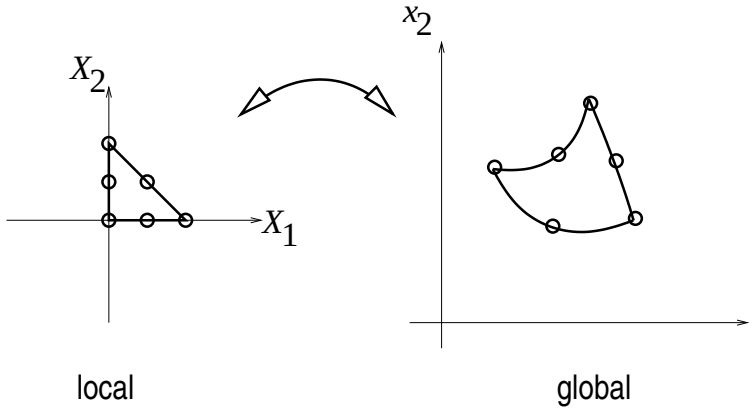


Figure 45: Isoparametric mapping of a P2 element.

From (112) or (113) it is easy to realize that the vertices are correctly mapped. onsider a vertex with local number s. Then $\tilde{\varphi}_s = 1$ at this vertex and zero at ıe others. This means that only one term in the sum is nonzero and $\boldsymbol{x} = \boldsymbol{x}_{q(e,s)}$, hich is the coordinate of this vertex in the global coordinate system.

.5 Computing integrals

et $\tilde{\Omega}^r$ denote the reference cell and $\Omega^{(e)}$ the cell in the physical coordinate ystem. The transformation of the integral from the physical to the reference oordinate system reads

$$\int_{\Omega^{(e)}} \varphi_i(\boldsymbol{x})\varphi_j(\boldsymbol{x})\,\mathrm{d}x = \int_{\tilde{\Omega}^r} \tilde{\varphi}_i(\boldsymbol{X})\tilde{\varphi}_j(\boldsymbol{X})\det J\,\mathrm{d}X, \tag{114}$$

$$\int_{\Omega^{(e)}} \varphi_i(\boldsymbol{x})f(\boldsymbol{x})\,\mathrm{d}x = \int_{\tilde{\Omega}^r} \tilde{\varphi}_i(\boldsymbol{X})f(\boldsymbol{x}(\boldsymbol{X}))\det J\,\mathrm{d}X, \tag{115}$$

here $\mathrm{d}x$ means the infinitesimal area element $dxdy$ in 2D and $dxdydz$ in 3D, ith a similar definition of $\mathrm{d}X$. The quantity $\det J$ is the determinant of the acobian of the mapping $\boldsymbol{x}(\boldsymbol{X})$. In 2D,

$$J = \begin{bmatrix} \frac{\partial x}{\partial X} & \frac{\partial x}{\partial Y} \\ \frac{\partial y}{\partial X} & \frac{\partial y}{\partial Y} \end{bmatrix}, \quad \det J = \frac{\partial x}{\partial X}\frac{\partial y}{\partial Y} - \frac{\partial x}{\partial Y}\frac{\partial y}{\partial X}. \tag{116}$$

/ith the affine mapping (112), $\det J = 2\Delta$, where Δ is the area or volume of ıe cell in the physical coordinate system.

emark. Observe that finite elements in 2D and 3D builds on the same *ideas* nd *concepts* as in 1D, but there is simply much more to compute because the ecific mathematical formulas in 2D and 3D are more complicated and the book eeping with dof maps also gets more complicated. The manual work is tedious, ngthy, and error-prone so automation by the computer is a must.

.0 Exercises

xercise 1: Linear algebra refresher I

ook up the topic of *vector space* in your favorite linear algebra book or search r the term at Wikipedia. Prove that vectors in the plane (a, b) form a vector pace by showing that all the axioms of a vector space are satisfied. Similarly, rove that all linear functions of the form $ax + b$ constitute a vector space, $, b \in \mathbb{R}$.

On the contrary, show that all quadratic functions of the form $1 + ax^2 + bx$ *not* constitute a vector space. Filename: `linalg1.pdf`.

Exercise 2: Linear algebra refresher II

As an extension of Exercise 1, check out the topic of *inner product spaces*. a possible inner product for the space of all linear functions of the form $a, b \in \mathbb{R}$. Show that this inner product satisfies the general requirements inner product in a vector space. Filename: `linalg2.pdf`.

Exercise 3: Approximate a three-dimensional vecto plane

Given $\boldsymbol{f} = (1, 1, 1)$ in $\mathbb{R}^3$, find the best approximation vector $\boldsymbol{u}$ in th spanned by the unit vectors $(1, 0)$ and $(0, 1)$. Repeat the calculations u vectors $(2, 1)$ and $(1, 2)$. Filename: `vec111_approx.pdf`.

Exercise 4: Approximate the exponential function by functions

Let V be a function space with basis functions x^i, $i = 0, 1, \ldots, N$. Find approximation to $f(x) = \exp(-x)$ on $\Omega = [0, 8]$ among all functions i $N = 2, 4, 6$. Illustrate the three approximations in three separate plots.

Hint. The exercise is easy to solve if you apply the `lest_squares` and c functions in the `approx1D.py` module.
Filename: `exp_powers.py`.

Exercise 5: Approximate the sine function by power tions

In this exercise we want to approximate the sine function by polyno order $N + 1$. Consider two bases:

$$V_1 = \{x, x^3, x^5, \ldots, x^{N-2}, x^N\},$$
$$V_2 = \{1, x, x^2, x^3, \ldots, x^N\}.$$

The basis V_1 is motivated by the fact that the Taylor polynomial approx to the sine function has only odd powers, while V_2 is motivated by the ass that also the even powers could improve the approximation in a least- setting.

Compute the best approximation to $f(x) = \sin(x)$ among all func V_1 and V_2 on two domains of increasing sizes: $\Omega_{1,k} = [0, k\pi]$, $k = 2, 3 \ldots$ $\Omega_{2,k} = [-k\pi/2, k\pi/2]$, $k = 2, 3 \ldots, 6$. Make plots for all combinations o Ω_1, Ω_2, $k = 2, 3, \ldots, 6$.

Add a plot of the N-th degree Taylor polynomial approximation around $x = 0$.

Hint. You can make a loop over V_1 and V_2, a loop over Ω_1 and Ω_2, and a loop ver k. Inside the loops, call the functions `least_squares` and `comparison_plot` om the `approx1D` module. $N = 9$ is a suggested value.
ilename: `sin_powers.py`.

Exercise 6: Approximate a steep function by sines

ind the best approximation of $f(x) = \tanh(s(x - \pi))$ on $[0, 2\pi]$ in the space V ith basis $\psi_i(x) = \sin((2i + 1)x)$, $i \in \mathcal{I}_s = \{0, \ldots, N\}$. Make a movie showing ow $u = \sum_{j \in \mathcal{I}_s} c_j \psi_j(x)$ approximates $f(x)$ as N grows. Choose s such that f is eep ($s = 20$ may be appropriate).

Hint. One may naively call the `least_squares_orth` and `comparison_plot` om the `approx1D` module in a loop and extend the basis with one new element a each pass. This approach implies a lot of recomputations. A more efficient rategy is to let `least_squares_orth` compute with only one basis function at time and accumulate the corresponding `u` in the total solution.
ilename: `tanh_sines_approx.py`.

emarks. Approximation of a discontinuous (or steep) $f(x)$ by sines, results a slow convergence and oscillatory behavior of the approximation close to the brupt changes in f. This is known as the Gibb's phenomenon[17].

Exercise 7: Animate the approximation of a steep function y sines

Iake a movie that shows how the approximation in Exercise 6 is improved as N grows. Illustrate a smooth case where $s = 0.5$ and a steep case where $s = 20$ a the $\tanh(s(x - \pi))$ function. Filename: `tanh_sines_approx_movie.py`.

Exercise 8: Fourier series as a least squares approximation

iven a function $f(x)$ on an interval $[0, L]$, look up the formula for the coefficients $_j$ and b_j in the Fourier series of f:

$$f(x) = a_0 + \sum_{j=1}^{\infty} a_j \cos\left(j\frac{\pi x}{L}\right) + \sum_{j=1}^{\infty} b_j \sin\left(j\frac{\pi x}{L}\right).$$

Let an infinite-dimensional vector space V have the basis functions $\cos j\frac{\pi x}{L}$ or $j = 0, 1, \ldots, \infty$ and $\sin j\frac{\pi x}{L}$ for $j = 1, \ldots, \infty$. Show that the least squares pproximation method from Section 2 leads to a linear system whose solution oincides with the standard formulas for the coefficients in a Fourier series of (x) (see also Section 2.7). You may choose

[17] `http://en.wikipedia.org/wiki/Gibbs_phenomenon`

$$\psi_{2i} = \cos\left(i\frac{\pi}{L}x\right), \quad \psi_{2i+1} = \sin\left(i\frac{\pi}{L}x\right),$$

for $i = 0, 1, \ldots, N \to \infty$.

Choose $f(x) = \tanh(s(x - \frac{1}{2}))$ on $\Omega = [0, 1]$, which is a smooth funct with considerable steepness around $x = 1/2$ as s grows in size. Calcu coefficients in the Fourier expansion by solving the linear system, arisi the least squares or Galerkin methods, by hand. Plot some truncated of the series together with $f(x)$ to show how the series expansion conve $s = 10$ and $s = 100$. Filename: `Fourier_approx.py`.

Exercise 9: Approximate a steep function by Lagrange nomials

Use interpolation/collocation with uniformly distributed points and Ch nodes to approximate

$$f(x) = -\tanh(s(x - \frac{1}{2})), \quad x \in [0, 1],$$

by Lagrange polynomials for $s = 10$ and $s = 100$, and $N = 3, 6, 9, 11$ separate plots of the approximation for each combination of s, poi (Chebyshev or uniform), and N. Filename: `tanh_Lagrange.py`.

Exercise 10: Define nodes and elements

Consider a domain $\Omega = [0, 2]$ divided into the three P2 elements $[0, 1]$, and $[1.2, 2]$.

For P1 and P2 elements, set up the list of coordinates and nodes and the numbers of the nodes that belong to each element (`elements` cases: 1) nodes and elements numbered from left to right, and 2) no elements numbered from right to left. Filename: `fe_numberings1.py`.

Exercise 11: Define vertices, cells, and dof maps

Repeat Exercise 10, but define the data structures `vertices`, `cells`, and instead of `nodes` and `elements`. Filename: `fe_numberings2.py`.

Exercise 12: Construct matrix sparsity patterns

Exercise 10 describes a element mesh with a total of five elements, but different element and node orderings. For each of the two orderings, 5×5 matrix and fill in the entries that will be nonzero.

Hint. A matrix entry (i, j) is nonzero if i and j are nodes in the same Filename: `fe_sparsity_pattern.pdf`.

Exercise 13: Perform symbolic finite element computations

erform symbolic calculations to find formulas for the coefficient matrix and ght-hand side when approximating $f(x) = \sin(x)$ on $\Omega = [0, \pi]$ by two P1 ements of size $\pi/2$. Solve the system and compare $u(\pi/2)$ with the exact value Filename: `sin_approx_P1.py`.

Exercise 14: Approximate a steep function by P1 and P2 lements

iven

$$f(x) = \tanh(s(x - \frac{1}{2}))$$

se the Galerkin or least squares method with finite elements to find an pproximate function $u(x)$. Choose $s = 40$ and try $N_e = 4, 8, 16$ P1 ele- ients and $N_e = 2, 4, 8$ P2 elements. Integrate $f\varphi_i$ numerically. Filename: `anh_fe_P1P2_approx.py`.

Exercise 15: Approximate a steep function by P3 and P4 lements

olve Exercise 14 using $N_e = 1, 2, 4$ P3 and P4 elements. How will a colloca- on/interpolation method work in this case with the same number of nodes? ilename: `tanh_fe_P3P4_approx.py`.

Exercise 16: Investigate the approximation error in finite lements

he theory (93) from Section 6.4 predicts that the error in the Pd approximation f a function should behave as h^{d+1}. Use experiments to verify this asymptotic ehavior (i.e., for small enough h). Choose two examples: $f(x) = Ae^{-\omega x}$ on $), 3/\omega]$ and $f(x) = A\sin(\omega x)$ on $\Omega = [0, 2\pi/\omega]$ for constants A and ω. What appens if you try $f(x) = \sqrt{x}$ on $[0, 1]$?

lint. Run a series of experiments: $(h_i, E_)$, $i = 0, \ldots, m$, where E_i is the L^2 orm of the error corresponding to element length h_i. Assume an error model $= Ch^r$ and compute r from two successive experiments:

$$r_i = \ln(E_{i+1}/E_i)/\ln(h_{i+1}/h_i), \quad i = 0, \ldots, m-1\,.$$

opefully, the sequence $r_0, \ldots, r_{m-1}$ converges to the true r, and r_{m-1} can be aken as an approximation to r.
ilename: `Asinwt_interpolation_error.py`.

Exercise 17: Approximate a step function by finite ele

Approximate the step function

$$f(x) = \begin{cases} 1 & x < 1/2, \\ 2 & x \geq 1/2 \end{cases}$$

by 2, 4, and 8 P1 and P2 elements. Compare approximations visually.

Hint. This f can also be expressed in terms of the Heaviside functio $f(x) = H(x - 1/2)$. Therefore, f can be defined by

```
f = sp.Heaviside(x -  sp.Rational(1,2))
```

making the `approximate` function in the `fe_approx1D.py` module an candidate to solve the problem. However, `sympy` does not handle s integration with this particular integrand, and the `approximate` functio problem when converting `f` to a Python function (for plotting) since `Hea` is not an available function in `numpy`. It is better to make special-purpo for this case or perform all calculations by hand.
Filename: `Heaviside_approx_P1P2.py`..

Exercise 18: 2D approximation with orthogonal func

Assume we have basis functions $\varphi_i(x, y)$ in 2D that are orthogonal su $(\varphi_i, \varphi_j) = 0$ when $i \neq j$. The function `least_squares` in the file `approx` will then spend much time on computing off-diagonal terms in the co matrix that we know are zero. To speed up the computations, make a `least_squares_orth` that utilizes the orthogonality among the basis fu Apply the function to approximate

$$f(x, y) = x(1 - x)y(1 - y)e^{-x-y}$$

on $\Omega = [0, 1] \times [0, 1]$ via basis functions

$$\varphi_i(x, y) = \sin(p\pi x)\sin(q\pi y), \quad i = qN_x + p\,.$$

Hint. Get ideas from the function `least_squares_orth` in Section file `approx1D.py`[19].
Filename: `approx2D_lsorth_sin.py`.

[18] `http://tinyurl.com/nm5587k/fem/fe_approx2D.py`
[19] `http://tinyurl.com/nm5587k/fem/fe_approx1D.py`

Exercise 19: Use the Trapezoidal rule and P1 elements

Consider approximation of some $f(x)$ on an interval Ω using the least squares or Galerkin methods with P1 elements. Derive the element matrix and vector using the Trapezoidal rule (101) for calculating integrals on the reference element. Assemble the contributions, assuming a uniform cell partitioning, and show that the resulting linear system has the form $c_i = f(x_i)$ for $i \in \mathcal{I}_s$. Filename: `e_P1_trapez.pdf`.

Problem 20: Compare P1 elements and interpolation

We shall approximate the function

$$f(x) = 1 + \epsilon \sin(2\pi n x), \quad x \in \Omega = [0, 1],$$

where $n \in \mathbb{Z}$ and $\epsilon \geq 0$.

) Sketch $f(x)$ and find the wave length of the function.

) We want to use N_P elements per wave length. Show that the number of elements is then nN_P.

) The critical quantity for accuracy is the number of elements per wave length, not the element size in itself. It therefore suffices to study an f with just one wave length in $\Omega = [0, 1]$. Set $\epsilon = 0.5$.

Run the least squares or projection/Galerkin method for $N_P = 2, 4, 8, 16, 32$. Compute the error $E = ||u - f||_{L^2}$.

Hint. Use the `fe_approx1D_numint` module to compute u and use the technique from Section 6.4 to compute the norm of the error.

) Repeat the set of experiments in the above point, but use interpolation/collocation based on the node points to compute $u(x)$ (recall that c_i is now simply (x_i)). Compute the error $E = ||u - f||_{L^2}$. Which method seems to be most accurate?
Filename: `P1_vs_interp.py`.

Exercise 21: Implement 3D computations with global basis functions

Extend the `approx2D.py`[20] code to 3D applying ideas from Section 8.4. Use a 3D generalization of the test problem in Section 8.3 to test the implementation. Filename: `approx3D.py`.

Exercise 22: Use Simpson's rule and P2 elements

Redo Exercise 19, but use P2 elements and Simpson's rule based on sampling the integrands at the nodes in the reference cell. Filename: `fe_P2_simpson.pdf`.

[20] `http://tinyurl.com/nm5587k/fem/approx2D.py`

11 Basic principles for approximating diff tial equations

The finite element method is a very flexible approach for solving partial diff equations. Its two most attractive features are the ease of handling dor complex shape in two and three dimensions and the ease of constructing order discretization methods. The finite element method is usually app discretization in space, and therefore spatial problems will be our focu coming sections. Extensions to time-dependent problems may, for insta finite difference approximations in time.

Before studying how finite element methods are used to tackle diff equation, we first look at how global basis functions and the least Galerkin, and collocation principles can be used to solve differential eq

11.1 Differential equation models

Let us consider an abstract differential equation for a function $u(x)$ variable, written as

$$\mathcal{L}(u) = 0, \quad x \in \Omega\,.$$

Here are a few examples on possible choices of $\mathcal{L}(u)$, of increasing comp

$$\mathcal{L}(u) = \frac{d^2u}{dx^2} - f(x),$$
$$\mathcal{L}(u) = \frac{d}{dx}\left(\alpha(x)\frac{du}{dx}\right) + f(x),$$
$$\mathcal{L}(u) = \frac{d}{dx}\left(\alpha(u)\frac{du}{dx}\right) - au + f(x),$$
$$\mathcal{L}(u) = \frac{d}{dx}\left(\alpha(u)\frac{du}{dx}\right) + f(u, x)\,.$$

Both $\alpha(x)$ and $f(x)$ are considered as specified functions, while a is a pr parameter. Differential equations corresponding to (118)-(119) arise in c phenomena, such as steady transport of heat in solids and flow of viscou between flat plates. The form (120) arises when transient diffusion phenomenon are discretized in time by finite differences. The equatic appear in chemical models when diffusion of a substance is combin chemical reactions. Also in biology, (121) plays an important role, l spreading of species and in models involving generation and propag electrical signals.

Let $\Omega = [0, L]$ be the domain in one space dimension. In additior differential equation, u must fulfill boundary conditions at the boundarie domain, $x = 0$ and $x = L$. When $\mathcal{L}$ contains up to second-order deriva

ı the examples above, $m = 1$, we need one boundary condition at each of the wo) boundary points, here abstractly specified as

$$\mathcal{B}_0(u) = 0,\ x = 0, \quad \mathcal{B}_1(u) = 0,\ x = L \tag{122}$$

There are three common choices of boundary conditions:

$$\mathcal{B}_i(u) = u - g, \qquad \text{Dirichlet condition} \tag{123}$$

$$\mathcal{B}_i(u) = -\alpha\frac{du}{dx} - g, \qquad \text{Neumann condition} \tag{124}$$

$$\mathcal{B}_i(u) = -\alpha\frac{du}{dx} - h(u - g), \qquad \text{Robin condition} \tag{125}$$

ere, g and a are specified quantities.

From now on we shall use $u_{\mathrm{e}}(x)$ as symbol for the *exact* solution, fulfilling

$$\mathcal{L}(u_{\mathrm{e}}) = 0, \quad x \in \Omega, \tag{126}$$

hile $u(x)$ is our notation for an *approximate* solution of the differential equation.

Remark on notation.

In the literature about the finite element method, is common to use u as the exact solution and u_h as the approximate solution, where h is a discretization parameter. However, the vast part of the present text is about the approximate solutions, and having a subscript h attached all the time is cumbersome. Of equal importance is the close correspondence between implementation and mathematics that we strive to achieve in this text: when it is natural to use `u` and not `u_h` in code, we let the mathematical notation be dictated by the code's preferred notation. After all, it is the powerful computer implementations of the finite element method that justifies studying the mathematical formulation and aspects of the method.

1.2 Simple model problems

common model problem used much in the forthcoming examples is

$$-u''(x) = f(x), \quad x \in \Omega = [0, L], \quad u(0) = 0,\ u(L) = D\,. \tag{127}$$

closely related problem with a different boundary condition at $x = 0$ reads

$$-u''(x) = f(x), \quad x \in \Omega = [0, L], \quad u'(0) = C,\ u(L) = D\,. \tag{128}$$

third variant has a variable coefficient,

$$-(\alpha(x)u'(x))' = f(x), \quad x \in \Omega = [0, L], \quad u'(0) = C,\ u(L) = D\,.$$

We can easily solve these using `sympy`. For (127) we can write the f

```
def model1(f, L, D):
    """Solve -u'' = f(x), u(0)=0, u(L)=D."""
    u_x = - sp.integrate(f, (x, 0, x)) + c_0
    u = sp.integrate(u_x, (x, 0, x)) + c_1
    r = sp.solve([u.subs(x, 0)-0, u.subs(x,L)-D], [c_0, c_1])
    u = u.subs(c_0, r[c_0]).subs(c_1, r[c_1])
    u = sp.simplify(sp.expand(u))
    return u
```

Calling `model1(2, L, D)` results in the solution

$$u(x) = \frac{1}{L}x\left(D + L^2 - Lx\right)$$

Model (128) can be solved by

```
def model2(f, L, C, D):
    """Solve -u'' = f(x), u'(0)=C, u(L)=D."""
    u_x = - sp.integrate(f, (x, 0, x)) + c_0
    u = sp.integrate(u_x, (x, 0, x)) + c_1
    r = sp.solve([sp.diff(u,x).subs(x, 0)-C, u.subs(x,L)-D], [c_0
    u = u.subs(c_0, r[c_0]).subs(c_1, r[c_1])
    u = sp.simplify(sp.expand(u))
    return u
```

to yield

$$u(x) = -x^2 + Cx - CL + D + L^2,$$

if $f(x) = 2$. Model (129) requires a bit more involved code,

```
def model3(f, a, L, C, D):
    """Solve -(a*u')' = f(x), u(0)=C, u(L)=D."""
    au_x = - sp.integrate(f, (x, 0, x)) + c_0
    u = sp.integrate(au_x/a, (x, 0, x)) + c_1
    r = sp.solve([u.subs(x, 0)-C, u.subs(x,L)-D], [c_0, c_1])
    u = u.subs(c_0, r[c_0]).subs(c_1, r[c_1])
    u = sp.simplify(sp.expand(u))
    return u
```

With $f(x) = 0$ and $\alpha(x) = 1 + x^2$ we get

$$u(x) = \frac{C\,\mathrm{atan}\,(L) - C\,\mathrm{atan}\,(x) + D\,\mathrm{atan}\,(x)}{\mathrm{atan}\,(L)}$$

1.3 Forming the residual

he fundamental idea is to seek an approximate solution u in some space V,

$$V = \text{span}\{\psi_0(x), \ldots, \psi_N(x)\},$$

hich means that u can always be expressed as a linear combination of the basis nctions $\{\varphi_i\}_{i\in\mathcal{I}_s}$, with $\mathcal{I}_s$ as the index set $\{0, \ldots, N\}$:

$$u(x) = \sum_{j\in\mathcal{I}_s} c_j\psi_j(x)\,.$$

he coefficients $\{c_i\}_{i\in\mathcal{I}_s}$ are unknowns to be computed.

(Later, in Section 14, we will see that if we specify boundary values of u ifferent from zero, we must look for an approximate solution $u(x) = B(x) + \sum_j c_j\psi_j(x)$, where $\sum_j c_j\psi_j \in V$ and $B(x)$ is some function for incorporating ıe right boundary values. Because of $B(x)$, u will not necessarily lie in V. This ıodification does not imply any difficulties.)

We need principles for deriving $N+1$ equations to determine the $N+1$ nknowns $\{c_i\}_{i\in\mathcal{I}_s}$. When approximating a given function f by $u = \sum_j c_j\varphi_j$, a ey idea is to minimize the square norm of the approximation error $e = u - f$ or quvalently) demand that e is orthogonal to V. Working with e is not so useful ere since the approximation error in our case is $e = u_{\text{e}} - u$ and u_{e} is unknown. he only general indicator we have on the quality of the approximate solution is o what degree u fulfills the differential equation. Inserting $u = \sum_j c_j\psi_j$ into (u) reveals that the result is not zero, because u is only likely to equal u_{e}. The onzero result,

$$R = \mathcal{L}(u) = \mathcal{L}(\sum_j c_j\psi_j), \tag{132}$$

called the *residual* and measures the error in fulfilling the governing equation.

Various principles for determining $\{c_i\}_{i\in\mathcal{I}_s}$ try to minimize R in some sense. ote that R varies with x and the $\{c_i\}_{i\in\mathcal{I}_s}$ parameters. We may write this ependence explicitly as

$$R = R(x; c_0, \ldots, c_N)\,. \tag{133}$$

elow, we present three principles for making R small: a least squares method, projection or Galerkin method, and a collocation or interpolation method.

1.4 The least squares method

he least-squares method aims to find $\{c_i\}_{i\in\mathcal{I}_s}$ such that the square norm of the esidual

$$||R|| = (R, R) = \int_\Omega R^2\,\mathrm{d}x \tag{134}$$

minimized. By introducing an inner product of two functions f and g on Ω as

$$(f, g) = \int_\Omega f(x)g(x)\,\mathrm{d}x,$$

the least-squares method can be defined as

$$\min_{c_0,\ldots,c_N} E = (R, R)\,.$$

Differentiating with respect to the free parameters $\{c_i\}_{i\in\mathcal{I}_s}$ gives the equations

$$\int_\Omega 2R\frac{\partial R}{\partial c_i}\,\mathrm{d}x = 0 \quad \Leftrightarrow \quad (R, \frac{\partial R}{\partial c_i}) = 0, \quad i \in \mathcal{I}_s\,.$$

11.5 The Galerkin method

The least-squares principle is equivalent to demanding the error to be ort to the space V when approximating a function f by $u \in V$. With a diff equation we do not know the true error so we must instead require the R to be orthogonal to V. This idea implies seeking $\{c_i\}_{i\in\mathcal{I}_s}$ such that

$$(R, v) = 0, \quad \forall v \in V\,.$$

This is the Galerkin method for differential equations.

This statement is equivalent to R being orthogonal to the $N+1$ basis fu only:

$$(R, \psi_i) = 0, \quad i \in \mathcal{I}_s,$$

resulting in $N+1$ equations for determining $\{c_i\}_{i\in\mathcal{I}_s}$.

11.6 The Method of Weighted Residuals

A generalization of the Galerkin method is to demand that R is ort to some space W, but not necessarily the same space as V where we s unknown function. This generalization is naturally called the *method of residuals*:

$$(R, v) = 0, \quad \forall v \in W\,.$$

If $\{w_0, \ldots, w_N\}$ is a basis for W, we can equivalently express the me weighted residuals as

$$(R, w_i) = 0, \quad i \in \mathcal{I}_s\,.$$

The result is $N+1$ equations for $\{c_i\}_{i\in\mathcal{I}_s}$.

The least-squares method can also be viewed as a weighted residual with $w_i = \partial R/\partial c_i$.

Variational formulation of the continuous problem.

Formulations like (140) (or (141)) and (138) (or (139)) are known as *variational formulations*. These equations are in this text primarily used for a numerical approximation $u \in V$, where V is a *finite-dimensional* space with dimension $N+1$. However, we may also let V be an *infinite-dimensional* space containing the exact solution $u_{\mathrm{e}}(x)$ such that also u_{e} fulfills the same variational formulation. The variational formulation is in that case a mathematical way of stating the problem and acts as an alternative to the usual formulation of a differential equation with initial and/or boundary conditions.

1.7 Test and Trial Functions

ı the context of the Galerkin method and the method of weighted residuals it is ɔmmon to use the name *trial function* for the approximate $u = \sum_j c_j\psi_j$. The ɔace containing the trial function is known as the *trial space*. The function v ıtering the orthogonality requirement in the Galerkin method and the method f weighted residuals is called *test function*, and so are the ψ_i or w_i functions ıat are used as weights in the inner products with the residual. The space here the test functions comes from is naturally called the *test space*.

We see that in the method of weighted residuals the test and trial spaces are ifferent and so are the test and trial functions. In the Galerkin method the test nd trial spaces are the same (so far).

Remark.

It may be subject to debate whether it is only the form of (140) or (138) after integration by parts, as explained in Section 11.10, that qualifies for the term variational formulation. The result after integration by parts is what is obtained after taking the *first variation* of an optimization problem, see Section 11.13. However, here we use variational formulation as a common term for formulations which, in contrast to the differential equation $R = 0$, instead demand that an average of R is zero: $(R, v) = 0$ for all v in some space.

1.8 The collocation method

he idea of the collocation method is to demand that R vanishes at $N+1$ elected points $x_0, \ldots, x_N$ in Ω:

$$R(x_i; c_0, \ldots, c_N) = 0, \quad i \in \mathcal{I}_s\,. \tag{142}$$

The collocation method can also be viewed as a method of weighted r with Dirac delta functions as weighting functions. Let $\delta(x - x_i)$ be th delta function centered around $x = x_i$ with the properties that $\delta(x - x_i)$ $x \neq x_i$ and

$$\int_\Omega f(x)\delta(x - x_i)\,\mathrm{d}x = f(x_i), \quad x_i \in \Omega\,.$$

Intuitively, we may think of $\delta(x - x_i)$ as a very peak-shaped function $x = x_i$ with integral 1, roughly visualized in Figure 46. Because of (143), let $w_i = \delta(x - x_i)$ be weighting functions in the method of weighted r and (141) becomes equivalent to (142).

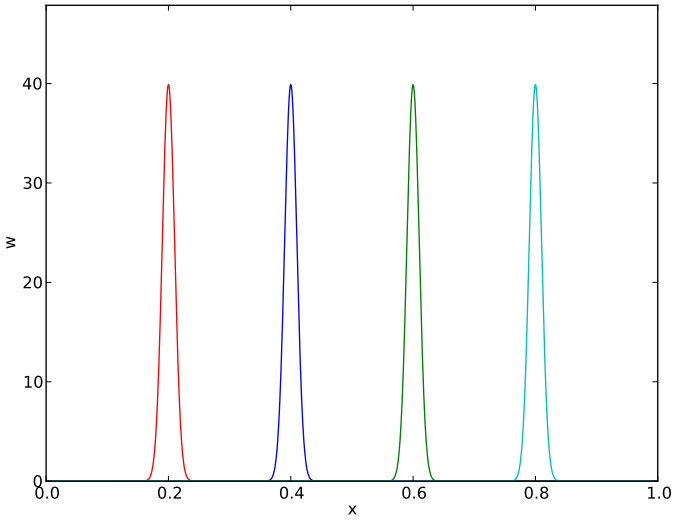


Figure 46: Approximation of delta functions by narrow Gaussian func

The subdomain collocation method. The idea of this approach mand the integral of R to vanish over $N+1$ subdomains Ω_i of Ω:

$$\int_{\Omega_i} R\,\mathrm{d}x = 0, \quad i \in \mathcal{I}_s\,.$$

This statement can also be expressed as a weighted residual method

$$\int_\Omega Rw_i\,\mathrm{d}x = 0, \quad i \in \mathcal{I}_s,$$

where $w_i = 1$ for $x \in \Omega_i$ and $w_i = 0$ otherwise.

1.9 Examples on using the principles

et us now apply global basis functions to illustrate the principles for minimizing
.

he model problem. We consider the differential equation problem

$$-u''(x) = f(x), \quad x \in \Omega = [0, L], \quad u(0) = 0,\ u(L) = 0\,. \tag{146}$$

asis functions. Our choice of basis functions ψ_i for V is

$$\psi_i(x) = \sin\left((i+1)\pi\frac{x}{L}\right), \quad i \in \mathcal{I}_s\,. \tag{147}$$

An important property of these functions is that $\psi_i(0) = \psi_i(L) = 0$, which eans that the boundary conditions on u are fulfilled:

$$u(0) = \sum_j c_j\psi_j(0) = 0, \quad u(L) = \sum_j c_j\psi_j(L) = 0\,.$$

nother nice property is that the chosen sine functions are orthogonal on Ω:

$$\int_0^L \sin\left((i+1)\pi\frac{x}{L}\right)\sin\left((j+1)\pi\frac{x}{L}\right)\,\mathrm{d}x = \begin{cases} \frac{1}{2}L & i = j \\ 0, & i \neq j \end{cases} \tag{148}$$

rovided i and j are integers.

he residual. We can readily calculate the following explicit expression for ie residual:

$$\begin{aligned} R(x; c_0, \ldots, c_N) &= u''(x) + f(x), \\ &= \frac{d^2}{dx^2}\left(\sum_{j\in\mathcal{I}_s} c_j\psi_j(x)\right) + f(x), \\ &= \sum_{j\in\mathcal{I}_s} c_j\psi_j''(x) + f(x)\,. \end{aligned} \tag{149}$$

he least squares method. The equations (137) in the least squares method equire an expression for $\partial R/\partial c_i$. We have

$$\frac{\partial R}{\partial c_i} = \frac{\partial}{\partial c_i}\left(\sum_{j\in\mathcal{I}_s} c_j\psi_j''(x) + f(x)\right) = \sum_{j\in\mathcal{I}_s}\frac{\partial c_j}{\partial c_i}\psi_j''(x) = \psi_i''(x)\,. \tag{150}$$

he governing equations for $\{c_i\}_{i\in\mathcal{I}_s}$ are then

$$(\sum_j c_j\psi_j'' + f, \psi_i'') = 0, \quad i \in \mathcal{I}_s,$$

which can be rearranged as

$$\sum_{j\in\mathcal{I}_s}(\psi_i'', \psi_j'')c_j = -(f, \psi_i''), \quad i \in \mathcal{I}_s\,.$$

This is nothing but a linear system

$$\sum_{j\in\mathcal{I}_s} A_{i,j}c_j = b_i, \quad i \in \mathcal{I}_s,$$

with

$$\begin{aligned} A_{i,j} &= (\psi_i'', \psi_j'') \\ &= \pi^4(i+1)^2(j+1)^2L^{-4}\int_0^L \sin\left((i+1)\pi\frac{x}{L}\right)\sin\left((j+1)\pi\frac{x}{L}\right) \\ &= \begin{cases} \frac{1}{2}L^{-3}\pi^4(i+1)^4 & i = j \\ 0, & i \neq j \end{cases} \\ b_i &= -(f, \psi_i'') = (i+1)^2\pi^2L^{-2}\int_0^L f(x)\sin\left((i+1)\pi\frac{x}{L}\right)\,\mathrm{d}x \end{aligned}$$

Since the coefficient matrix is diagonal we can easily solve for

$$c_i = \frac{2L}{\pi^2(i+1)^2}\int_0^L f(x)\sin\left((i+1)\pi\frac{x}{L}\right)\,\mathrm{d}x\,.$$

With the special choice of $f(x) = 2$ can be calculated in `sympy` by

```
from sympy import *
import sys

i, j = symbols('i j', integer=True)
x, L = symbols('x L')
f = 2
a = 2*L/(pi**2*(i+1)**2)
c_i = a*integrate(f*sin((i+1)*pi*x/L), (x, 0, L))
c_i = simplify(c_i)
print c_i
```

The answer becomes

$$c_i = 4\frac{L^2\left((-1)^i + 1\right)}{\pi^3\left(i^3 + 3i^2 + 3i + 1\right)}$$

Now, $1 + (-1)^i = 0$ for i odd, so only the coefficients with even index are Introducing $i = 2k$ for $k = 0, \ldots, N/2$ to count the relevant indices (for k goes to $(N-1)/2$), we get the solution

$$u(x) = \sum_{k=0}^{N/2} \frac{8L^2}{\pi^3(2k+1)^3} \sin\left((2k+1)\pi\frac{x}{L}\right) . \tag{156}$$

he coefficients decay very fast: $c_2 = c_0/27$, $c_4 = c_0/125$. The solution will herefore be dominated by the first term,

$$u(x) \approx \frac{8L^2}{\pi^3} \sin\left(\pi\frac{x}{L}\right) .$$

he Galerkin method. The Galerkin principle (138) applied to (146) consists f inserting our special residual (149) in (138)

$$(u'' + f, v) = 0, \quad \forall v \in V,$$

r

$$(u'', v) = -(f, v), \quad \forall v \in V . \tag{157}$$

his is the variational formulation, based on the Galerkin principle, of our ifferential equation. The $\forall v \in V$ requirement is equivalent to demanding the quation $(u'', v) = -(f, v)$ to be fulfilled for all basis functions $v = \psi_i$, $i \in \mathcal{I}_s$, e (138) and (139). We therefore have

$$(\sum_{j\in\mathcal{I}_s} c_j\psi_j'', \psi_i) = -(f, \psi_i), \quad i \in \mathcal{I}_s . \tag{158}$$

his equation can be rearranged to a form that explicitly shows that we get a near system for the unknowns $\{c_i\}_{i\in\mathcal{I}_s}$:

$$\sum_{j\in\mathcal{I}_s} (\psi_i, \psi_j'')c_j = (f, \psi_i), \quad i \in \mathcal{I}_s . \tag{159}$$

or the particular choice of the basis functions (147) we get in fact the same near system as in the least squares method because $\psi'' = -(i+1)^2\pi^2L^{-2}\psi$.

he collocation method. For the collocation method (142) we need to decide pon a set of $N+1$ collocation points in Ω. A simple choice is to use uniformly paced points: $x_i = i\Delta x$, where $\Delta x = L/N$ in our case ($N \geq 1$). However, these oints lead to at least two rows in the matrix consisting of zeros (since $\psi_i(x_0) = 0$ nd $\psi_i(x_N) = 0$), thereby making the matrix singular and non-invertible. This rces us to choose some other collocation points, e.g., random points or points niformly distributed in the interior of Ω. Demanding the residual to vanish at ese points leads, in our model problem (146), to the equations

$$-\sum_{j\in\mathcal{I}_s} c_j\psi_j''(x_i) = f(x_i), \quad i \in \mathcal{I}_s, \tag{160}$$

hich is seen to be a linear system with entries

$$A_{i,j} = -\psi_j''(x_i) = (j+1)^2\pi^2L^{-2} \sin\left((j+1)\pi\frac{x_i}{L}\right),$$

in the coefficient matrix and entries $b_i = 2$ for the right-hand side $f(x) = 2$).

The special case of $N = 0$ can sometimes be of interest. A natural c then the midpoint $x_0 = L/2$ of the domain, resulting in $A_{0,0} = -\psi_0''(x_0) = f(x_0) = 2$, and hence $c_0 = 2L^2/\pi^2$.

Comparison. In the present model problem, with $f(x) = 2$, the exact is $u(x) = x(L-x)$, while for $N = 0$ the Galerkin and least squares result in $u(x) = 8L^2\pi^{-3}\sin(\pi x/L)$ and the collocation method leads to $2L^2\pi^{-2}\sin(\pi x/L)$. Since all methods fulfill the boundary conditions $u(L) = 0$, we expect the largest discrepancy to occur at the midpoin domain: $x = L/2$. The error at the midpoint becomes $-0.008L^2$ for the G and least squares method, and $0.047L^2$ for the collocation method.

11.10 Integration by parts

A problem arises if we want to apply popular finite element functions our model problem (146) by the standard least squares, Galerkin, or col methods: the piecewise polynomials $\psi_i(x)$ have discontinuous deriva the cell boundaries which makes it problematic to compute the secon derivative. This fact actually makes the least squares and collocation meth suitable for finite element approximation of the unknown function. (By re the equation $-u'' = f$ as a system of two first-order equations, $u' = v$ an f, the least squares method can be applied. Also, differentiating discon functions can actually be handled by distribution theory in mathematic Galerkin method and the method of weighted residuals can, however, be together with finite element basis functions if we use *integration by pa* means for transforming a second-order derivative to a first-order one.

Consider the model problem (146) and its Galerkin formulation

$$-(u'', v) = (f, v) \quad \forall v \in V .$$

Using integration by parts in the Galerkin method, we can move a deriv u onto v:

$$\begin{aligned}\int_0^L u''(x)v(x)\,\mathrm{d}x &= -\int_0^L u'(x)v'(x)\,\mathrm{d}x + [vu']_0^L \\ &= -\int_0^L u'(x)v'(x)\,\mathrm{d}x + u'(L)v(L) - u'(0)v(0) .\end{aligned}$$

Usually, one integrates the problem at the stage where the u and v fu enter the formulation. Alternatively, but less common, we can integrate l in the expressions for the matrix entries:

$$\int_0^L \psi_i(x)\psi_j''(x)\,\mathrm{d}x = -\int_0^L \psi_i'(x)\psi_j'(x)dx + [\psi_i\psi_j']_0^L$$
$$= -\int_0^L \psi_i'(x)\psi_j'(x)\,\mathrm{d}x + \psi_i(L)\psi_j'(L) - \psi_i(0)\psi_j'(0)\,. \quad (162)$$

ıtegration by parts serves to reduce the order of the derivatives and to make the ɔefficient matrix symmetric since $(\psi_i', \psi_j') = (\psi_i', \psi_j')$. The symmetry property epends on the type of terms that enter the differential equation. As will be seen ıter in Section 15, integration by parts also provides a method for implementing oundary conditions involving u'.

With the choice (147) of basis functions we see that the "boundary terms" $_i(L)\psi_j'(L)$ and $\psi_i(0)\psi_j'(0)$ vanish since $\psi_i(0) = \psi_i(L) = 0$.

Veak form. Since the variational formulation after integration by parts make eaker demands on the differentiability of u and the basis functions ψ_i, the result- ıg integral formulation is referred to as a *weak form* of the differential equation roblem. The original variational formulation with second-order derivatives, or ıe differential equation problem with second-order derivative, is then the *strong ɔrm*, with stronger requirements on the differentiability of the functions.

For differential equations with second-order derivatives, expressed as varia- onal formulations and solved by finite element methods, we will always perform ıtegration by parts to arrive at expressions involving only first-order derivatives.

1.11 Boundary function

ɔ far we have assumed zero Dirichlet boundary conditions, typically $u(0) = (L) = 0$, and we have demanded that $\psi_i(0) = \psi_i(L) = 0$ for $i \in \mathcal{I}_s$. What bout a boundary condition like $u(L) = D \neq 0$? This condition immediately ıces a problem: $u = \sum_j c_j\varphi_j(L) = 0$ since all $\varphi_i(L) = 0$.

A boundary condition of the form $u(L) = D$ can be implemented by demand- ıg that all $\psi_i(L) = 0$, but adding a *boundary function* $B(x)$ with the right oundary value, $B(L) = D$, to the expansion for u:

$$u(x) = B(x) + \sum_{j\in\mathcal{I}_s} c_j\psi_j(x)\,.$$

his u gets the right value at $x = L$:

$$u(L) = B(L) + \sum_{j\in\mathcal{I}_s} c_j\psi_j(L) = B(L) = D\,.$$

he idea is that for any boundary where u is known we demand ψ_i to vanish nd construct a function $B(x)$ to attain the boundary value of u. There are ɔ restrictions how $B(x)$ varies with x in the interior of the domain, so this ariation needs to be constructed in some way.

For example, with $u(0) = 0$ and $u(L) = D$, we can choose $B(x) =$ since this form ensures that $B(x)$ fulfills the boundary conditions: $B(0)$ $B(L) = D$. The unknown function is then sought on the form

$$u(x) = \frac{x}{L}D + \sum_{j\in\mathcal{I}_s} c_j\psi_j(x),$$

with $\psi_i(0) = \psi_i(L) = 0$.

The $B(x)$ function can be chosen in many ways as long as its boundar are correct. For example, $B(x) = D(x/L)^p$ for any power p will work fin above example.

As another example, consider a domain $\Omega = [a, b]$ where the bc conditions are $u(a) = U_a$ and $u(b) = U_b$. A class of possible $B(x)$ funct

$$B(x) = U_a + \frac{U_b - U_a}{(b-a)^p}(x-a)^p, \quad p > 0\,.$$

Real applications will most likely use the simplest version, $p = 1$, but h a p parameter was included to demonstrate the ambiguity in the constru $B(x)$.

Summary.

The general procedure of incorporating Dirichlet boundary conditions as follows. Let $\partial\Omega_E$ be the part(s) of the boundary $\partial\Omega$ of the dom where u is specified. Set $\psi_i = 0$ at the points in $\partial\Omega_E$ and seek u as

$$u(x) = B(x) + \sum_{j\in\mathcal{I}_s} c_j\psi_j(x),$$

where $B(x)$ equals the boundary conditions on u at $\partial\Omega_E$.

Remark. With the $B(x)$ term, u does not in general lie in $V = \mathrm{span}\,\{\psi_0$ anymore. Moreover, when a prescribed value of u at the boundary, say $u($ is different from zero, it does not make sense to say that u lies in a vecto because this space does not obey the requirements of addition and sca tiplication. For example, $2u$ does not lie in the space since its boundar is $2U_a$, which is incorrect. It only makes sense to split u in two parts, above, and have the unknown part $\sum_j c_j\psi_j$ in a proper function space.

11.12 Abstract notation for variational formulation

We have seen that variational formulations end up with a formula involvir v, such as (u', v') and a formula involving v and known functions, such a A widely used notation is to introduce an abstract variational statement as $a(u, v) = L(v)$, where $a(u, v)$ is a so-called *bilinear form* involving

erms that contain both the test and trial function, while $L(v)$ is a *linear form* ontaining all the terms without the trial function. For example, the statement

$$\int_\Omega u'v'\,\mathrm{d}x = \int_\Omega fv\,\mathrm{d}x \quad \text{or} \quad (u', v') = (f, v) \quad \forall v \in V$$

an be written in abstract form: *find u such that*

$$a(u, v) = L(v) \quad \forall v \in V,$$

here we have the definitions

$$a(u, v) = (u', v'), \quad L(v) = (f, v)\,.$$

The term *linear* means that $L(\alpha_1 v_1 + \alpha_2 v_2) = \alpha_1 L(v_1) + \alpha_2 L(v_2)$ for two est functions v_1 and v_2, and scalar parameters α_1 and α_2. Similarly, the term *linear* means that $a(u, v)$ is linear in both its arguments:

$$a(\alpha_1 u_1 + \alpha_2 u_2, v) = \alpha_1 a(u_1, v) + \alpha_2 a(u_2, v),$$
$$a(u, \alpha_1 v_1 + \alpha_2 v_2) = \alpha_1 a(u, v_1) + \alpha_2 a(u, v_2)\,.$$

n nonlinear problems these linearity properties do not hold in general and the bstract notation is then $F(u; v) = 0$.

The matrix system associated with $a(u, v) = L(v)$ can also be written in an bstract form by inserting $v = \psi_i$ and $u = \sum_j c_j\psi_j$ in $a(u, v) = L(v)$. Using the near properties, we get

$$\sum_{j\in\mathcal{I}_s} a(\psi_j, \psi_i)c_j = L(\psi_i), \quad i \in \mathcal{I}_s,$$

hich is a linear system

$$\sum_{j\in\mathcal{I}_s} A_{i,j}c_j = b_i, \quad i \in \mathcal{I}_s,$$

here

$$A_{i,j} = a(\psi_j, \psi_i), \quad b_i = L(\psi_i)\,.$$

n many problems, $a(u, v)$ is symmetric such that $a(\psi_j, \psi_i) = a(\psi_i, \psi_j)$. In those ases the coefficient matrix becomes symmetric, $A_{i,j} = A_{j,i}$, a property that can mplify solution algorithms for linear systems and make them more stable in ddition to saving memory and computations.

The abstract notation $a(u, v) = L(v)$ for linear differential equation problems much used in the literature and in description of finite element software n particular the FEniCS[21] documentation). We shall frequently summarize ariational forms using this notation.

[21] `http://fenicsproject.org`

11.13 Variational problems and optimization of fun als

If $a(u, v) = a(v, u)$, it can be shown that the variational statement

$$a(u, v) = L(v) \quad \forall v \in V,$$

is equivalent to minimizing the functional

$$F(v) = \frac{1}{2}a(v, v) - L(v)$$

over all functions $v \in V$. That is,

$$F(u) \leq F(v) \quad \forall v \in V\,.$$

Inserting a $v = \sum_j c_j\psi_j$ turns minimization of $F(v)$ into minimizati quadratic function

$$\bar{F}(c_0, \ldots, c_N) = \sum_{j\in\mathcal{I}_s}\sum_{i\in\mathcal{I}_s} a(\psi_i, \psi_j)c_i c_j - \sum_{j\in\mathcal{I}_s} L(\psi_j)c_j$$

of $N + 1$ parameters.

Minimization of $\bar{F}$ implies

$$\frac{\partial\bar{F}}{\partial c_i} = 0, \quad i \in \mathcal{I}_s\,.$$

After some algebra one finds

$$\sum j \in \mathcal{I}_s a(\psi_i, \psi_j)c_j = L(\psi_i), \quad i \in \mathcal{I}_s,$$

which is the same system as that arising from $a(u, v) = L(v)$.

Many traditional applications of the finite element method, especially mechanics and structural analysis, start with formulating $F(v)$ from p principles, such as minimization of energy, and then proceeds with $a(u, v) = L(v)$, which is the equation usually desired in implementation

12 Examples on variational formulations

The following sections derive variational formulations for some prototyp ential equations in 1D, and demonstrate how we with ease can handle coefficients, mixed Dirichlet and Neumann boundary conditions, fir derivatives, and nonlinearities.

2.1 Variable coefficient

onsider the problem

$$-\frac{d}{dx}\left(\alpha(x)\frac{du}{dx}\right) = f(x), \quad x\in\Omega=[0,L],\ u(0)=C,\ u(L)=D\,. \tag{166}$$

here are two new features of this problem compared with previous examples: variable coefficient $a(x)$ and nonzero Dirichlet conditions at both boundary oints.

Let us first deal with the boundary conditions. We seek

$$u(x) = B(x) + \sum_{j\in\mathcal{I}_s} c_j\psi_i(x),$$

ith $\psi_i(0)=\psi_i(L)=0$ for $i\in\mathcal{I}_s$. The function $B(x)$ must then fulfill $B(0)=C$ nd $B(L)=D$. How B varies in between $x=0$ and $x=L$ is not of importance. ne possible choice is

$$B(x) = C + \frac{1}{L}(D-C)x,$$

hich follows from (164) with $p=1$.

We seek $(u-B)\in V$. As usual,

$$V = \text{span}\{\psi_0,\ldots,\psi_N\},$$

ut the two Dirichlet boundary conditions demand that

$$\psi_i(0)=\psi_i(L)=0, \quad i\in\mathcal{I}_s\,.$$

ote that any $v\in V$ has the property $v(0)=v(L)=0$.

The residual arises by inserting our u in the differential equation:

$$R = -\frac{d}{dx}\left(\alpha\frac{du}{dx}\right) - f\,.$$

alerkin's method is

$$(R,v)=0, \quad \forall v\in V,$$

r written with explicit integrals,

$$\int_\Omega \left(\frac{d}{dx}\left(\alpha\frac{du}{dx}\right) - f\right) v\,\mathrm{d}x = 0, \quad \forall v\in V\,.$$

/e proceed with integration by parts to lower the derivative from second to first rder:

$$-\int_\Omega \frac{d}{dx}\left(\alpha(x)\frac{du}{dx}\right) v\,\mathrm{d}x = \int_\Omega \alpha(x)\frac{du}{dx}\frac{dv}{dx}\,\mathrm{d}x - \left[\alpha\frac{du}{dx}v\right]_0^L\,.$$

The boundary term vanishes since $v(0)=v(L)=0$. The variational tion is then

$$\int_\Omega \alpha(x)\frac{du}{dx}\frac{dv}{dx}\,\mathrm{d}x = \int_\Omega f(x)v\,\mathrm{d}x, \quad \forall v\in V\,.$$

The variational formulation can alternatively be written in a more compa

$$(\alpha u', v') = (f,v), \quad \forall v\in V\,.$$

The corresponding abstract notation reads

$$a(u,v) = L(v) \quad \forall v\in V,$$

with

$$a(u,v) = (\alpha u', v'), \quad L(v) = (f,v)\,.$$

Note that the a in the notation $a(\cdot,\cdot)$ is not to be mixed with the coefficient $a(x)$ in the differential equation.

We may insert $u = B + \sum_j c_j\psi_j$ and $v=\psi_i$ to derive the linear sys

$$(\alpha B' + \alpha\sum_{j\in\mathcal{I}_s} c_j\psi_j', \psi_i') = (f,\psi_i), \quad i\in\mathcal{I}_s\,.$$

Isolating everything with the c_j coefficients on the left-hand side and al terms on the right-hand side gives

$$\sum_{j\in\mathcal{I}_s} (\alpha\psi_j', \psi_i')c_j = (f,\psi_i) + (a(D-C)L^{-1},\psi_i'), \quad i\in\mathcal{I}_s\,.$$

This is nothing but a linear system $\sum_j A_{i,j}c_j = b_i$ with

$$A_{i,j} = (a\psi_j',\psi_i') = \int_\Omega \alpha(x)\psi_j'(x),\psi_i'(x)\,\mathrm{d}x,$$

$$b_i = (f,\psi_i) + (a(D-C)L^{-1},\psi_i') = \int_\Omega \left(f(x)\psi_i(x) + \alpha(x)\frac{D-C}{L}\psi_i'(x\right.$$

12.2 First-order derivative in the equation and bou condition

The next problem to formulate in variational form reads

$$-u''(x) + bu'(x) = f(x), \quad x\in\Omega=[0,L],\ u(0)=C,\ u'(L)=E\,.$$

The new features are a first-order derivative u' in the equation and the b condition involving the derivative: $u'(L)=E$. Since we have a Dirichlet c at $x=0$, we must force $\psi_i(0)=0$ and use a boundary function to take the condition $u(0)=C$. Because there is no Dirichlet condition on $x=$

ot make any requirements to $\psi_i(L)$. The simplest possible choice of $B(x)$ is $B(x) = C$.

The expansion for u becomes

$$u = C + \sum_{j\in\mathcal{I}_s} c_j\psi_i(x)\,.$$

The variational formulation arises from multiplying the equation by a test unction $v \in V$ and integrating over Ω:

$$(-u'' + bu' - f, v) = 0, \quad \forall v \in V$$

We apply integration by parts to the $u''v$ term only. Although we could also ntegrate $u'v$ by parts, this is not common. The result becomes

$$(u' + bu', v') = (f, v) + [u'v]_0^L, \quad \forall v \in V\,.$$

Now, $v(0) = 0$ so

$$[u'v]_0^L = u'(L)v(L) = Ev(L),$$

because $u'(L) = E$. Integration by parts allows us to take care of the Neumann ondition in the boundary term.

Natural and essential boundary conditions.

Omitting a boundary term like $[u'v]_0^L$ implies that we actually impose the condition $u' = 0$ unless there is a Dirichlet condition (i.e., $v = 0$) at that point! This result has great practical consequences, because it is easy to forget the boundary term, and this mistake may implicitly set a boundary condition! Since homogeneous Neumann conditions can be incorporated without doing anything, and non-homogeneous Neumann conditions can just be inserted in the boundary term, such conditions are known as *natural boundary conditions*. Dirichlet conditions requires more essential steps in the mathematical formulation, such as forcing all $\varphi_i = 0$ on the boundary and constructing a $B(x)$, and are therefore known as *essential boundary conditions*.

The final variational form reads

$$(u', v') + (bu', v) = (f, v) + Ev(L), \quad \forall v \in V\,.$$

n the abstract notation we have

$$a(u, v) = L(v) \quad \forall v \in V,$$

ith the particular formulas

$$a(u, v) = (u', v') + (bu', v), \quad L(v) = (f, v) + Ev(L)\,.$$

The associated linear system is derived by inserting $u = B + \sum_j c_j$ replacing v by ψ_i for $i \in \mathcal{I}_s$. Some algebra results in

$$\sum_{j\in\mathcal{I}_s} \underbrace{((\psi_j', \psi_i') + (b\psi_j', \psi_i))}_{A_{i,j}} c_j = \underbrace{(f, \psi_i) + E\psi_i(L)}_{b_i}\,.$$

Observe that in this problem, the coefficient matrix is not symmetric, of the term

$$(b\psi_j', \psi_i) = \int_\Omega b\psi_j'\psi_i\,\mathrm{d}x \neq \int_\Omega b\psi_i'\psi_j\,\mathrm{d}x = (\psi_i', b\psi_j)\,.$$

12.3 Nonlinear coefficient

Finally, we show that the techniques used above to derive variationa also apply to nonlinear differential equation problems as well. Here is a problem with a nonlinear coefficient and right-hand side:

$$-(\alpha(u)u')' = f(u), \quad x \in [0, L],\ u(0) = 0,\ u'(L) = E\,.$$

Our space V has basis $\{\psi_i\}_{i\in\mathcal{I}_s}$, and because of the condition $u(0) = 0$, require $\psi_i(0) = 0$, $i \in \mathcal{I}_s$.

Galerkin's method is about inserting the approximate u, multiply differential equation by $v \in V$, and integrate,

$$-\int_0^L \frac{d}{dx}\left(\alpha(u)\frac{du}{dx}\right) v\,\mathrm{d}x = \int_0^L f(u)v\,\mathrm{d}x \quad \forall v \in V\,.$$

The integration by parts does not differ from the case where we have $\alpha(x)$ of $\alpha(u)$:

$$\int_0^L \alpha(u)\frac{du}{dx}\frac{dv}{dx}\,\mathrm{d}x = \int_0^L f(u)v\,\mathrm{d}x + [\alpha(u)vu']_0^L \quad \forall v \in V\,.$$

The term $\alpha(u(0))v(0)u'(0) = 0$ since $v(0)$. The other term, $\alpha(u(L))v(L)$ used to impose the other boundary condition $u'(L) = E$, resulting in

$$\int_0^L \alpha(u)\frac{du}{dx}\frac{dv}{dx}v\,\mathrm{d}x = \int_0^L f(u)v\,\mathrm{d}x + \alpha(u(L))v(L)E \quad \forall v \in V,$$

or alternatively written more compactly as

$$(\alpha(u)u', v') = (f(u), v) + \alpha(L)v(L)E \quad \forall v \in V\,.$$

Since the problem is nonlinear, we cannot identify a bilinear form $a(u, v)$ linear form $L(v)$. An abstract notation is typically *find u such that*

$$F(u; v) = 0 \quad \forall v \in V,$$

ith

$$F(u;v) = (a(u)u', v') - (f(u), v) - a(L)v(L)E\,.$$

By inserting $u = \sum_j c_j\psi_j$ we get a *nonlinear system of algebraic equations* or the unknowns c_i, $i \in \mathcal{I}_s$. Such systems must be solved by constructing a equence of linear systems whose solutions hopefully converge to the solution f the nonlinear system. Frequently applied methods are Picard iteration and ewton's method.

2.4 Computing with Dirichlet and Neumann conditions

et us perform the necessary calculations to solve

$$-u''(x) = 2, \quad x \in \Omega = [0,1], \quad u'(0) = C,\ u(1) = D,$$

sing a global polynomial basis $\psi_i \sim x^i$. The requirements on ψ_i is that $\psi_i(1) = 0$, ecause u is specified at $x = 1$, so a proper set of polynomial basis functions can e

$$\psi_i(x) = (1-x)^{i+1}, \quad i \in \mathcal{I}_s\,.$$

suitable $B(x)$ function to handle the boundary condition $u(1) = D$ is $B(x) = x$. The variational formulation becomes

$$(u', v') = (2, v) - Cv(0) \quad \forall v \in V\,.$$

he entries in the linear system are then

$$A_{i,j} = (\psi_j, \psi_i) = \int_0^1 \psi_i'(x)\psi_j'(x)\,\mathrm{d}x = \int_0^1 (i+1)(j+1)(1-x)^{i+j}\,\mathrm{d}x = \frac{ij+i+j+1}{i+j+1}$$

$$\begin{aligned} b_i &= (2, \psi_i) - (D, \psi_i') - C\psi_i(0) \\ &= \int_0^1 (2\psi_i(x) - D\psi_i'(x))\,\mathrm{d}x - C\psi_i(0) \\ &= \int_0^1 \left(2(1-x)^{i+1} - D(i+1)(1-x)^i\right)\,\mathrm{d}x - C\psi_i(0) \\ &= \frac{2-(2+i)(D+C)}{i+2}\,. \end{aligned}$$

With $N = 1$ the global matrix system is

$$\begin{pmatrix} 1 & 1 \\ 1 & 4/3 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} -C + D + 1 \\ 2/3 - C + D \end{pmatrix}$$

he solution becomes $c_0 = -C + D + 2$ and $c_1 = -1$, resulting in

$$u(x) = 1 - x^2 + D + C(x-1), \tag{169}$$

The exact solution is found by. integrating twice and applying the b conditions, either by hand or using `sympy` as shown in Section 11.2. It that the numerical solution coincides with the exact one. This result expected because if $(u_e - B) \in V$, $u = u_e$, as proved next.

12.5 When the numerical method is exact

We have some variational formulation: find $(u - B) \in V$ such that a $L(u)$ $\forall V$. The exact solution also fulfills $a(u_e, v) = L(v)$, but normally (lies in a much larger (infinite-dimensional) space. Suppose, neverthele $u_e = B + E$, where $E \in V$. That is, apart from Dirichlet conditions, u_e our finite-dimensional space V we use to compute u. Writing also u on t form $u = B + F$, we have

$$\begin{aligned} a(B+E, v) &= L(v) \quad \forall v \in V, \\ a(B+F, v) &= L(v) \quad \forall v \in V\,. \end{aligned}$$

Subtracting the equations show that $a(E - F, v) = 0$ for all $v \in V$, and t $E - F = 0$ and $u = u_e$.

The case treated in Section 12.4 is of the type where $u_e - B$ is a qu function that is 0 at $x = 1$, and therefore $(u_e - B) \in V$, and the meth the exact solution.

13 Computing with finite elements

The purpose of this section is to demonstrate in detail how the finite method can the be applied to the model problem

$$-u''(x) = 2, \quad x \in (0, L),\ u(0) = u(L) = 0,$$

with variational formulation

$$(u', v') = (2, v) \quad \forall v \in V\,.$$

The variational formulation is derived in Section 11.10.

13.1 Finite element mesh and basis functions

We introduce a finite element mesh with N_e cells, all with length h, and the cells from left to right. global nodes. Choosing P1 elements, there nodes per cell, and the coordinates of the nodes become

$$x_i = ih, \quad h = L/N_e, \quad i = 0, \ldots, N_n = N_e + 1,$$

provided we number the nodes from left to right.

Each of the nodes, i, is associated a finite element basis function $\varphi_i(x)$. When pproximating a given function f by a finite element function u, we expand using finite element basis functions associated with *all* nodes in the mesh, e., $N = N_n$. However, when solving differential equations we will often have $I < N_n$ because of Dirichlet boundary conditions. Why this is the case will ow be explained in detail.

In our case with homogeneous Dirichlet boundary conditions we do not need ny boundary function $B(x)$ and can work with the expansion

$$u(x) = \sum_{j\in\mathcal{I}_s} c_j\psi_j(x)\,. \tag{170}$$

ecause of the boundary conditions, we must demand $\psi_i(0) = \psi_i(L) = 0$, $i \in \mathcal{I}_s$. /hen ψ_i, $i = 0,\ldots,N$, is to be selected among the finite element basis functions $_j$, $i = 0,\ldots,N_n$, we have to avoid using φ_j functions that do not vanish at $_0 = 0$ and $x_{N_n} = L$. However, all φ_j vanish at these two nodes for $j = 1,\ldots,N_n$. nly basis functions associated with the end nodes, φ_0 and φ_{N_n}, violate the oundary conditions of our differential equation. Therefore, we select the basis nctions φ_i to be the set of finite element basis functions associated with all ıe interior nodes in the mesh:

$$\psi_i = \varphi_{i+1}, \quad i = 0,\ldots,N\,.$$

ere, $N = N_n - 2$.

In the general case, the nodes are not necessarily numbered from left to right, o we introduce a mapping from the node numbering, or more precisely the egree of freedom numbering, to the numbering of the unknowns in the final quation system. These unknowns take on the numbers $0,\ldots,N$. Unknown umber j in the linear system corresponds to degree of freedom number $\nu(j)$, $\in \mathcal{I}_s$. We can then write

$$\psi_i = \varphi_{\nu(i)}, \quad i = 0,\ldots,N\,.$$

/ith a regular numbering as in the present example, $\nu(j) = j+1$, $j = 1,\ldots,N = _n - 2$.

3.2 Computation in the global physical domain

/e shall first perform a computation in the x coordinate system because the ıtegrals can be easily computed here by simple, visual, geometric considerations. his is called a global approach since we work in the x coordinate system and ompute integrals on the global domain $[0, L]$.

The entries in the coefficient matrix and right-hand side are

$$A_{i,j} = \int_0^L \psi_i'(x)\psi_j'(x)\,\mathrm{d}x, \quad b_i = \int_0^L 2\psi_i(x)\,\mathrm{d}x, \quad i,j \in \mathcal{I}_s\,.$$

xpressed in terms of finite element basis functions φ_i we get the alternative xpressions

$$A_{i,j} = \int_0^L \varphi_{i+1}'(x)\varphi_{j+1}'(x)\,\mathrm{d}x, \quad b_i = \int_0^L 2\varphi_{i+1}(x)\,\mathrm{d}x, \quad i,j \in \mathcal{I}_s$$

For the following calculations the subscripts on the finite element basis fu are more conveniently written as i and j instead of $i+1$ and $j+1$ notation becomes

$$A_{i-1,j-1} = \int_0^L \varphi_i'(x)\varphi_j'(x)\,\mathrm{d}x, \quad b_{i-1} = \int_0^L 2\varphi_i(x)\,\mathrm{d}x,$$

where the i and j indices run as $i,j = 1,\ldots,N_n - 1 = N + 1$.

The $\varphi_i(x)$ function is a hat function with peak at $x = x_i$ and a linear v in $[x_{i-1}, x_i]$ and $[x_i, x_{i+1}]$. The derivative is $1/h$ to the left of x_i and - the right, or more formally,

$$\varphi_i'(x) = \begin{cases} 0, & x < x_{i-1}, \\ h^{-1}, & x_{i-1} \le x < x_i, \\ -h^{-1}, & x_i \le x < x_{i+1}, \\ 0, & x \ge x_{i+1} \end{cases}$$

Figure 47 shows $\varphi_1'(x)$ and $\varphi_2'(x)$.

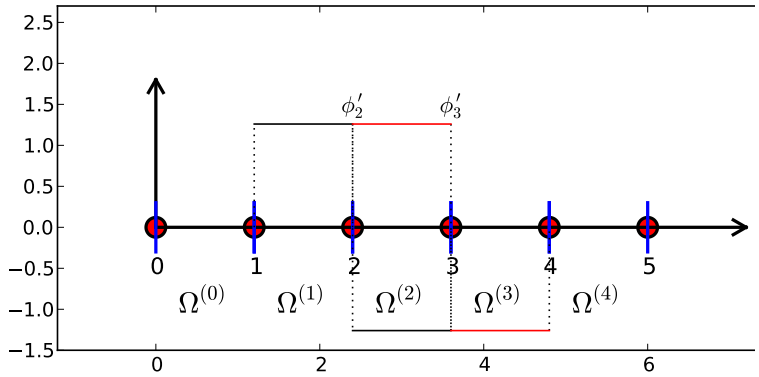


Figure 47: Illustration of the derivative of piecewise linear basis fu associated with nodes in cell 2.

We realize that φ_i' and φ_j' has no overlap, and hence their prod ishes, unless i and j are nodes belonging to the same cell. The only contributions to the coefficient matrix are therefore

$$A_{i-1,i-2} = \int_0^L \varphi_i'(x)\varphi_{i-1}'(x)\,\mathrm{d}x,$$
$$A_{i-1,i-1} = \int_0^L \varphi_i'(x)^2\,\mathrm{d}x,$$
$$A_{i-1,i} = \int_0^L \varphi_i'(x)\varphi_{i+1}'(x)\,\mathrm{d}x,$$

or $i = 1, \ldots, N_n - 1$, but for $i = 1$, $A_{i-1,i-2}$ is not defined, and for $i = N_n - 1$, $A_{i-1,i}$ is not defined.

We see that $\varphi_{i-1}'(x)$ and $\varphi_i'(x)$ have overlap of one cell $\Omega^{(i-1)} = [x_{i-1}, x_i]$ and that their product then is $-1/h^2$. The integrand is constant and therefore $A_{i-1,i-2} = -h^{-2}h = -h^{-1}$. A similar reasoning can be applied to $A_{i-1,i}$, which also becomes $-h^{-1}$. The integral of $\varphi_i'(x)^2$ gets contributions from two cells, $\Omega^{(i-1)} = [x_{i-1}, x_i]$ and $\Omega^{(i)} = [x_i, x_{i+1}]$, but $\varphi_i'(x)^2 = h^{-2}$ in both cells, and the length of the integration interval is $2h$ so we get $A_{i-1,i-1} = 2h^{-1}$.

The right-hand side involves an integral of $2\varphi_i(x)$, $i = 1, \ldots, N_n - 1$, which is just the area under a hat function of height 1 and width $2h$, i.e., equal to h. Hence, $b_{i-1} = 2h$.

To summarize the linear system, we switch from i to $i+1$ such that we can write

$$A_{i,i-1} = A_{i,i-1} = -h^{-1}, \quad A_{i,i} = 2h^{-1}, \quad b_i = 2h\,.$$

The equation system to be solved only involves the unknowns c_i for $i \in \mathcal{I}_s$. With our numbering of unknowns and nodes, we have that c_i equals $u(x_{i+1})$. The complete matrix system that takes the following form:

$$\frac{1}{h}\begin{pmatrix} 2 & -1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ -1 & 2 & -1 & \ddots & & & & & \vdots \\ 0 & -1 & 2 & -1 & \ddots & & & & \vdots \\ \vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & 0 & -1 & 2 & -1 & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_N \end{pmatrix} = \begin{pmatrix} 2h \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 2h \end{pmatrix} \tag{172}$$

3.3 Comparison with a finite difference discretization

A typical row in the matrix system can be written as

$$-\frac{1}{h}c_{i-1} + \frac{2}{h}c_i - \frac{1}{h}c_{i+1} = 2h\,.$$

Let us introduce the notation u_j for the value of u at node j: $u_j = u(x$ we have the interpretation $u(x_j) = \sum_j c_j\varphi(x_j) = \sum_j c_j\delta_{ij} = c_j$. The ur $c_0, \ldots, c_N$ are $u_1, \ldots, u_{N_n}$. Shifting i with $i+1$ in (173) and inserting u_i we get

$$-\frac{1}{h}u_{i-1} + \frac{2}{h}u_i - \frac{1}{h}u_{i+1} = 2h,$$

A finite difference discretization of $-u''(x) = 2$ by a centered, secor finite difference approximation $u''(x_i) \approx [D_xD_xu]_i$ with $\Delta x = h$ yields

$$-\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} = 2,$$

which is, in fact, equivalent to (174) if (174) is divided by h. Theref finite difference and the finite element method are equivalent in this sir problem.

Sometimes a finite element method generates the finite difference ec on a uniform mesh, and sometimes the finite element method generates ec that are different. The differences are modest, but may influence the nu quality of the solution significantly, especially in time-dependent proble

13.4 Cellwise computations

We now employ the cell by cell computational procedure where an elemen and vector are calculated for each cell and assembled in the global linear All integrals are mapped to the local reference coordinate system $X \in$ In the present case, the matrix entries contain derivatives with respect

$$A^{(e)}_{i-1,j-1} = \int_{\Omega^{(e)}} \varphi_i'(x)\varphi_j'(x)\,\mathrm{d}x = \int_{-1}^{1} \frac{d}{dx}\tilde{\varphi}_r(X)\frac{d}{dx}\tilde{\varphi}_s(X)\frac{h}{2}\,\mathrm{d}X,$$

where the global degree of freedom i is related to the local degree of fre through $i = q(e, r)$. Similarly, $j = q(e, s)$. The local degrees of freedom $r, s = 0, 1$ for a P1 element.

The integral for the element matrix. There are simple formulas basis functions $\tilde{\varphi}_r(X)$ as functions of X. However, we now need to derivative of $\tilde{\varphi}_r(X)$ with respect to x. Given

$$\tilde{\varphi}_0(X) = \frac{1}{2}(1 - X), \quad \tilde{\varphi}_1(X) = \frac{1}{2}(1 + X),$$

we can easily compute $d\tilde{\varphi}_r/dX$:

$$\frac{d\tilde{\varphi}_0}{dX} = -\frac{1}{2}, \quad \frac{d\tilde{\varphi}_1}{dX} = \frac{1}{2}\,.$$

rom the chain rule,

$$\frac{d\tilde{\varphi}_r}{dx} = \frac{d\tilde{\varphi}_r}{dX}\frac{dX}{dx} = \frac{2}{h}\frac{d\tilde{\varphi}_r}{dX}\,. \tag{176}$$

he transformed integral is then

$$A^{(e)}_{i-1,j-1} = \int_{\Omega^{(e)}} \varphi_i'(x)\varphi_j'(x)\,\mathrm{d}x = \int_{-1}^{1} \frac{2}{h}\frac{d\tilde{\varphi}_r}{dX}\frac{2}{h}\frac{d\tilde{\varphi}_s}{dX}\frac{h}{2}\,\mathrm{d}X\,.$$

'he integral for the element vector. The right-hand side is transformed ccording to

$$b^{(e)}_{i-1} = \int_{\Omega^{(e)}} 2\varphi_i(x)\,\mathrm{d}x = \int_{-1}^{1} 2\tilde{\varphi}_r(X)\frac{h}{2}\,\mathrm{d}X, \quad i = q(e,r),\ r = 0,1\,.$$

)etailed calculations of the element matrix and vector. Specifically or P1 elements we arrive at the following calculations for the element matrix ıtries:

$$\begin{aligned}
\tilde{A}^{(e)}_{0,0} &= \int_{-1}^{1} \frac{2}{h}\left(-\frac{1}{2}\right)\frac{2}{h}\left(-\frac{1}{2}\right)\frac{2}{h}\,\mathrm{d}X = \frac{1}{h}\\
\tilde{A}^{(e)}_{0,1} &= \int_{-1}^{1} \frac{2}{h}\left(-\frac{1}{2}\right)\frac{2}{h}\left(\frac{1}{2}\right)\frac{2}{h}\,\mathrm{d}X = -\frac{1}{h}\\
\tilde{A}^{(e)}_{1,0} &= \int_{-1}^{1} \frac{2}{h}\left(\frac{1}{2}\right)\frac{2}{h}\left(-\frac{1}{2}\right)\frac{2}{h}\,\mathrm{d}X = -\frac{1}{h}\\
\tilde{A}^{(e)}_{1,1} &= \int_{-1}^{1} \frac{2}{h}\left(\frac{1}{2}\right)\frac{2}{h}\left(\frac{1}{2}\right)\frac{2}{h}\,\mathrm{d}X = \frac{1}{h}
\end{aligned}$$

he element vector entries become

$$\begin{aligned}
\tilde{b}^{(e)}_0 &= \int_{-1}^{1} 2\frac{1}{2}(1-X)\frac{h}{2}\,\mathrm{d}X = h\\
\tilde{b}^{(e)}_1 &= \int_{-1}^{1} 2\frac{1}{2}(1+X)\frac{h}{2}\,\mathrm{d}X = h\,.
\end{aligned}$$

xpressing these entries in matrix and vector notation, we have

$$\tilde{A}^{(e)} = \frac{1}{h}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \quad \tilde{b}^{(e)} = h\begin{pmatrix} 1 \\ 1 \end{pmatrix}. \tag{177}$$

Contributions from the first and last cell. The first and last cell only one unknown and one basis function because of the Dirichlet bc conditions at the first and last node. The element matrix therefore be 1×1 matrix and there is only one entry in the element vector. On cell $\psi_0 = \varphi_1$ is involved, corresponding to integration with $\tilde{\varphi}_1$. On cell N $\psi_N = \varphi_{N_n-1}$ is involved, corresponding to integration with $\tilde{\varphi}_0$. We then special end-cell contributions

$$\tilde{A}^{(e)} = \frac{1}{h}\begin{pmatrix} 1 \end{pmatrix}, \quad \tilde{b}^{(e)} = h\begin{pmatrix} 1 \end{pmatrix},$$

for $e = 0$ and $e = N_e$. In these cells, we have only one degree of freed two as in the interior cells.

Assembly. The next step is to assemble the contributions from the cells. The assembly of an element matrix and vector into the global mat right-hand side can be expressed as

$$A_{q(e,r),q(e,s)} = A_{q(e,r),q(e,s)} + \tilde{A}^{(e)}_{r,s}, \quad b_{q(e,r)} = b_{q(e,r)} + \tilde{b}^{(e)}_r,$$

for r and s running over all local degrees of freedom in cell e.

To make the assembly algorithm more precise, it is convenient to Python data structures and a code snippet for carrying out all detail algorithm. For a mesh of four equal-sized P1 elements and $L = 2$ we ha

```
vertices = [0, 0.5, 1, 1.5, 2]
cells = [[0, 1], [1, 2], [2, 3], [3, 4]]
dof_map = [[0], [0, 1], [1, 2], [2]]
```

The total number of degrees of freedom is 3, being the function value internal 3 nodes where u is unknown. In cell 0 we have global degree of 0, the next cell has u unknown at its two nodes, which become global de freedom 0 and 1, and so forth according to the `dof_map` list. The mathe $q(e,r)$ quantity is nothing but the `dof_map` list.

Assume all element matrices are stored in a list `Ae` such that `Ae[e]` $\tilde{A}^{(e)}_{i,j}$. A corresponding list for the element vectors is named `be`, where `be` is $\tilde{b}^{(e)}_r$. A Python code snippet illustrates all details of the assembly alg

```
# A[i,j]: coefficient matrix, b[i]: right-hand side
for e in range(len(Ae)):
    for r in range(Ae[e].shape[0]):
        for s in range(Ae[e].shape[1]):
            A[dof_map[e,r],dof_map[e,s]] += Ae[e][i,j]
        b[dof_map[e,r]] += be[e][i,j]
```

The general case with `N_e` P1 elements of length `h` has

```
N_n = N_e + 1
vertices = [i*h for i in range(N_n)]
cells = [[e, e+1] for e in range(N_e)]
dof_map = [[0]] + [[e-1, e] for i in range(1, N_e)] + [[N_n-2]]
```

Carrying out the assembly results in a linear system that is identical to (172), hich is not surprising since the procedures is mathematically equivalent to the alculations in the physical domain.

A fundamental problem with the matrix system we have assembled is that ıe boundary conditions are not incorporated if $u(0)$ or $u(L)$ are different from ero. The next sections deals with this issue.

.4 Boundary conditions: specified nonzero value

Ve have to take special actions to incorporate Dirichlet conditions, such as $(L) = D$, into the computational procedures. The present section outlines lternative, yet mathematically equivalent, methods.

4.1 General construction of a boundary function

ı Section 11.11 we introduce a boundary function $B(x)$ to deal with nonzero irichlet boundary conditions for u. The construction of such a function is not lways trivial, especially not in multiple dimensions. However, a simple and eneral construction idea exists when the basis functions have the property

$$\varphi_i(x_j) = \delta_{ij}, \quad \delta_{ij} = \begin{cases} 1, & i = j, \\ 0, & i \neq j, \end{cases}$$

here x_j is a boundary point. Examples on such functions are the Lagrange ıterpolating polynomials and finite element functions.

Suppose now that u has Dirichlet boundary conditions at nodes with numbers $\in I_b$. For example, $I_b = \{0, N_n\}$ in a 1D mesh with node numbering from left right. Let U_i be the corresponding prescribed values of $u(x_i)$. We can then, ı general, use

$$B(x) = \sum_{j \in I_b} U_j \varphi_j(x)\,. \tag{179}$$

is easy to verify that $B(x_i) = \sum_{j \in I_b} U_j \varphi_j(x_i) = U_i$.

The unknown function can then be written as

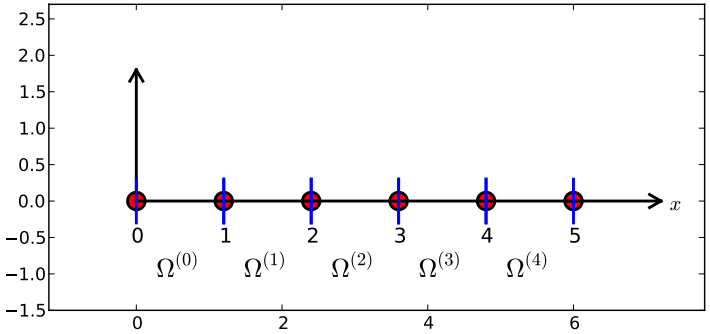
$$u(x) = \sum_{j \in I_b} U_j \varphi_j(x) + \sum_{j \in \mathcal{I}_s} c_j \varphi_{\nu(j)}, \tag{180}$$

here $\nu(j)$ maps unknown number j in the equation system to node $\nu(j)$. We an easily show that with this u, a Dirichlet condition $u(x_k) = U_k$ is fulfilled:

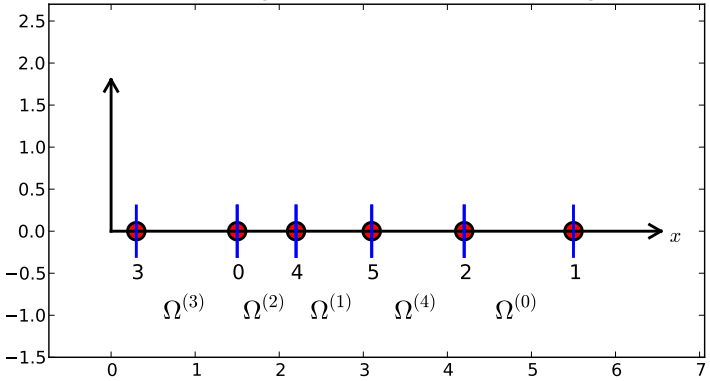
$$u(x_k) = \sum_{j \in I_b} U_j \underbrace{\varphi_j(x)}_{\neq 0 \text{ only for } j=k} + \sum_{j \in \mathcal{I}_s} c_j \underbrace{\varphi_{\nu(j)}(x_k)}_{=0,\ k \notin \mathcal{I}_s} = U_k$$

Some examples will further clarify the notation. With a regular left-numbering of nodes in a mesh with P1 elements, and Dirichlet condi $x = 0$, we use finite element basis functions associated with the nodes $1, 2,$ implying that $\nu(j) = j + 1$, $j = 0, \ldots, N$, where $N = N_n - 1$. For the pa mesh below the expansion becomes

$$u(x) = U_0\varphi_0(x) + c_0\varphi_1(x) + c_1\varphi_2(x) + \cdots + c_4\varphi_5(x)\,.$$



Here is a mesh with an irregular cell and node numbering:



Say we in this latter mesh have Dirichlet conditions on the left-m right-most node, with numbers 3 and 1, respectively. Then we can nun unknowns at the interior nodes from left to right, giving $\nu(0) = 0$, ν $\nu(2) = 5$, $\nu(3) = 2$. This gives

$$B(x) = U_3\varphi_3(x) + U_1\varphi_1(x),$$

and

$$u(x) = B(x) + \sum_{j=0}^{3} c_j \varphi_{\nu(j)} = U_3\varphi_3 + U_1\varphi_1 + c_0\varphi_0 + c_1\varphi_4 + c_2\varphi_5 + c_3\varphi_2\,.$$

Switching to the more standard case of left-to-right numbering and boundary onditions $u(0) = C$, $u(L) = D$, we have $N = N_n - 2$ and

$$\begin{aligned} u(x) &= C\varphi_0 + D\varphi_{N_n} + \sum_{j\in\mathcal{I}_s} c_j\varphi_{j+1} \\ &= C\varphi_0 + D\varphi_{N_n} + c_0\varphi_1 + c_1\varphi_2 + \cdots + c_N\varphi_{N_n-1}\,. \end{aligned}$$

The idea of constructing B described here generalizes almost trivially to 2D nd 3D problems: $B = \sum_{j\in I_b} U_j\varphi_j$, where I_b is the index set containing the umbers of all the nodes on the boundaries where Dirichlet values are prescribed.

4.2 Example on computing with finite element-based a boundary function

et us see how the model problem $-u'' = 2$, $u(0) = C$, $u(L) = D$, is affected by $B(x)$ to incorporate boundary values. Inserting the expression

$$u(x) = B(x) + \sum_{j\in\mathcal{I}_s} c_j\psi_j(x)$$

ı $-(u'', \psi_i) = (f, \psi_i)$ and integrating by parts results in a linear system with

$$A_{i,j} = \int_0^L \psi_i'(x)\psi_j'(x)\,\mathrm{d}x, \quad b_i = \int_0^L (f(x) - B'(x))\psi_i(x)\,\mathrm{d}x\,.$$

/e choose $\psi_i = \varphi_{i+1}$, $i = 0, \ldots, N = N_n - 2$ if the node numbering is from left › right. (Later we also need the assumption that the cells too are numbered om left to right.) The boundary function becomes

$$B(x) = C\varphi_0(x) + D\varphi_{N_n}(x)\,.$$

he expansion for $u(x)$ is

$$u(x) = B(x) + \sum_{j\in\mathcal{I}_s} c_j\varphi_{j+1}(x)\,.$$

/e can write the matrix and right-hand side entries as

$$_{i-1,j-1} = \int_0^L \varphi_i'(x)\varphi_j'(x)\,\mathrm{d}x, \quad b_{i-1} = \int_0^L (f(x) - C\varphi_0'(x) - D\varphi_{N_n}'(x))\varphi_i(x)\,\mathrm{d}x,$$

›r $i, j = 1, \ldots, N+1 = N_n - 1$. Note that we have here used $B' = C\varphi_0' + D\varphi_{N_n}'$.

Computations in physical coordinates. Most of the terms in th system have already been computed so we concentrate on the new cont from the boundary function. The integral $C\int_0^L \varphi_0'(x))\varphi_i(x)\,\mathrm{d}x$ can on nonzero contribution from the first cell, $\Omega^{(0)} = [x_0, x_1]$ since $\varphi_0'(x) =$ other cells. Moreover, $\varphi_0'(x)\varphi_i(x)\,\mathrm{d}x \neq 0$ only for $i = 0$ and $i = 1$ (but excluded), since $\varphi_i = 0$ on the first cell if $i > 1$. With a similar reaso realize that $D\int_0^L \varphi_{N_n}'(x))\varphi_i(x)\,\mathrm{d}x$ can only get a nonzero contribution f last cell. From the explanations of the calculations in Section 3.6 we tl that

$$\int_0^L \varphi_0'(x)\varphi_1(x)\,\mathrm{d}x = \frac{1}{h}\cdot\frac{1}{h} = -\frac{1}{2}, \quad \int_0^L \varphi_{N_n}'(x)\varphi_{N_n-1}(x)\,\mathrm{d}x = \frac{1}{h}\cdot\frac{1}{h}$$

The extra boundary term because of $B(x)$ boils down to adding $C/2$ tc $-D/2$ to b_N.

Cellwise computations on the reference element. As an equiv alternative, we now turn to cellwise computations. The element matri vectors are calculated as Section 13.4, so we concentrate on the impac new term involving $B(x)$. We observe that $C\varphi_0' = 0$ on all cells excep and $D\varphi_{N_n}' = 0$ on all cells except $e = N_e$. In this case there is only one u in these cells since $u(0)$ and $u(L)$ are prescribed, so the element vector l one entry. The entry for the last cell, $e = N_e$, becomes

$$\tilde{b}_0^{(e)} = \int_{-1}^1 \left(f - D\frac{2}{h}\frac{d\tilde{\varphi}_1}{dX}\right)\tilde{\varphi}_0\frac{h}{2}\,\mathrm{d}X = (\frac{h}{2}(2 - D\frac{2}{h}\frac{1}{2})\int_{-1}^1 \tilde{\varphi}_0\,\mathrm{d}X = h -$$

Similar computations on the first cell yield

$$\tilde{b}_0^{(0)} = \int_{-1}^1 \left(f - C\frac{2}{h}\frac{d\tilde{\varphi}_0}{dX}\right)\tilde{\varphi}_1\frac{h}{2}\,\mathrm{d}X = (\frac{h}{2}(2 + C\frac{2}{h}\frac{1}{2})\int_{-1}^1 \tilde{\varphi}_1\,\mathrm{d}X = h +$$

When assembling these contributions, we see that b_0 gets right-hand sid linear system gets an extra term $C/2$ and b_N gets $-D/2$, as in the comp in the physical domain.

14.3 Modification of the linear system

From an implementational point of view, there is a convenient altern adding the $B(x)$ function and using only the basis functions associat nodes where u is truly unknown. Instead of seeking

$$u(x) = \sum_{j\in I_b} U_j\varphi_j(x) + \sum_{j\in\mathcal{I}_s} c_j\varphi_{\nu(j)}(x),$$

e use the sum over all degrees of freedom, including the known boundary values:

$$u(x) = \sum_{j \in \mathcal{I}_s} c_j \varphi_j(x)\,. \tag{182}$$

ote that the collections of unknowns $\{c_i\}_{i\in\mathcal{I}_s}$ in (181) and (182) are different: ı (181) N counts the number of nodes where u is not known, while in (181) N ɔunts all the nodes ($N = N_n$).

The idea is to compute the entries in the linear system as if no Dirichlet ılues are prescribed. Afterwards, we modify the linear system to ensure that ıe known c_j values are incorporated.

A potential problem arises for the boundary term $[u'v]_0^L$ from the integration y parts: imagining no Dirichlet conditions means that we no longer require $= 0$ at Dirichlet points, and the boundary term is then nonzero at these oints. However, when we modify the linear system, we will erase whatever the ɔntribution from $[u'v]_0^L$ should be at the Dirichlet points in the right-hand side f the linear system. We can therefore safely forget $[u'v]_0^L$ at any point where a ɔirichlet condition applies.

Computations in the physical system. Let us redo the computations in ıe example in Section 14.1. We solve $-u'' = 2$ with $u(0) = 0$ and $u(L) = D$. he expressions for $A_{i,j}$ and b_i are the same, but the numbering is different as ıe numbering of unknowns and nodes now coincide:

$$A_{i,j} = \int_0^L \varphi_i'(x)\varphi_j'(x)\,\mathrm{d}x, \quad b_i = \int_0^L f(x)\varphi_i(x)\,\mathrm{d}x,$$

ɔr $i, j = 0, \ldots, N = N_n$. The integrals involving basis functions corresponding ɔ interior mesh nodes, $i, j = 1, \ldots, N_n - 1$, are obviously the same as before. Ve concentrate on the contributions from φ_0 and φ_{N_n}:

$$\begin{aligned}
A_{0,0} &= \int_0^L (\varphi_0')^2\,\mathrm{d}x = \int_0^{x_1} = (\varphi_0')^2\,\mathrm{d}x\frac{1}{h},\\
A_{0,1} &= \int_0^L \varphi_0'\varphi_1'\,\mathrm{d}x = \int_0^{x_1} \varphi_0'\varphi_1'\,\mathrm{d}x = -\frac{1}{h},\\
A_{N,N} &= \int_0^L (\varphi_0')^2\,\mathrm{d}x = \int_{x_{N_n-1}}^{x_{N_n}} (\varphi_0')^2\,\mathrm{d}x = \frac{1}{h},\\
A_{N,N-1} &= \int_0^L (\varphi_0')^2\,\mathrm{d}x = \int_{x_{N_n-1}}^{x_{N_n}} (\varphi_0')^2\,\mathrm{d}x = -\frac{1}{h}\,.
\end{aligned}$$

The new terms on the right-hand side are also those involving φ_0 and φ_{N_n}:

$$\begin{aligned}
b_0 &= \int_0^L 2\varphi_0(x)\,\mathrm{d}x = \int_0^{x_1} 2\varphi_0(x)\,\mathrm{d}x = h,\\
b_N &= \int_0^L 2\varphi_{N_n}\,\mathrm{d}x = \int_{x_{N_n-1}}^{x_{N_n}} 2\varphi_{N_n}\,\mathrm{d}x = h\,.
\end{aligned}$$

The complete matrix system, involving all degrees of freedom, takes t

$$\frac{1}{h}\begin{pmatrix}
1 & -1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0\\
-1 & 2 & -1 & \ddots & & & & & \vdots\\
0 & -1 & 2 & -1 & \ddots & & & & \vdots\\
\vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots\\
\vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots\\
\vdots & & & 0 & -1 & 2 & -1 & \ddots & \vdots\\
\vdots & & & & \ddots & \ddots & \ddots & \ddots & 0\\
\vdots & & & & & \ddots & \ddots & \ddots & -1\\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & -1 & 1
\end{pmatrix}\begin{pmatrix} c_0\\ \vdots\\ \vdots\\ \vdots\\ \vdots\\ \vdots\\ \vdots\\ \vdots\\ c_N \end{pmatrix} = \begin{pmatrix}$$

Incorporation of Dirichlet values can now be done by replacing the f last equation by $c_0 = 0$ and $c_N = D$. This action changes the system tc

$$\frac{1}{h}\begin{pmatrix}
h & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0\\
-1 & 2 & -1 & \ddots & & & & & \vdots\\
0 & -1 & 2 & -1 & \ddots & & & & \vdots\\
\vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots\\
\vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots\\
\vdots & & & 0 & -1 & 2 & -1 & \ddots & \vdots\\
\vdots & & & & \ddots & \ddots & \ddots & \ddots & 0\\
\vdots & & & & & \ddots & \ddots & \ddots & -1\\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 & h
\end{pmatrix}\begin{pmatrix} c_0\\ \vdots\\ \vdots\\ \vdots\\ \vdots\\ \vdots\\ \vdots\\ \vdots\\ c_N \end{pmatrix} = \begin{pmatrix}$$

Note that because we do not require $\varphi_i(0) = 0$ and $\varphi_i(L)$, $i \in \mathcal{I}_s$, the b term $[u'v]_0^L$ gives in principle contributions $u'(0)\varphi_0(0)$ to b_0 and $u'(L)\varphi$ b_N ($u'\varphi_i$ vanishes for $x = 0$ or $x = L$ for $i = 1, \ldots, N-1$). Neverthe erase these contributions in b_0 and b_N and insert boundary values instea argument shows why we can drop computing $[u'v]_0^L$ at Dirichlet nodes v implement the Dirichlet values by modifying the linear system.

4.4 Symmetric modification of the linear system

he original matrix system (172) is symmetric, but the modifications in (184) estroy the symmetry. Our described modification will in general destroy an itial symmetry in the matrix system. This is not a particular computational isadvantage for tridiagonal systems arising in 1D problems, but may be more erious in 2D and 3D problems when the systems are large and exploiting ymmetry can be important for halving the storage demands, speeding up omputations, and/or making the solution algorithm more robust. Therefore, n alternative modification which preserves symmetry is frequently applied.

Let c_k be a coefficient corresponding to a known value $u(x_k) = U_k$. We ant to replace equation k in the system by $c_k = U_k$, i.e., insert zeroes in row umber k in the coefficient matrix, set 1 on the diagonal, and replace b_k by U_k. symmetry-preserving modification consists in first subtracting column number in the coefficient matrix, i.e., $A_{i,k}$ for $i \in \mathcal{I}_s$, times the boundary value U_k, om the right-hand side: $b_i \leftarrow b_i - A_{i,k}U_k$. Then we put zeroes in row number *and* column number k in the coefficient matrix, and finally set $b_k = U_k$. The eps in algorithmic form becomes

1. $b_i \leftarrow b_i - A_{i,k}U_k$ for $i \in \mathcal{I}_s$
2. $A_{i,k} = A_{k,i} = 0$ for $i \in \mathcal{I}_s$
3. $A_{k,k} = 1$
4. $b_i = U_k$

his modification goes as follows for the specific linear system written out in 83) in Section 14.3. First we subtract the first column in the coefficient matrix, mes the boundary value, from the right-hand side. Because $c_0 = 0$, this ıbtraction has no effect. Then we subtract the last column, times the boundary alue D, from the right-hand side. This action results in $b_{N-1} = 2h + D/h$ nd $b_N = h - 2D/h$. Thereafter, we place zeros in the first and last row and olumn in the coefficient matrix and 1 on the two corresponding diagonal entries. inally, we set $b_0 = 0$ and $b_N = D$. The result becomes

$$\frac{1}{h}\begin{pmatrix} h & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 2 & -1 & \ddots & & & & & \vdots \\ 0 & -1 & 2 & -1 & \ddots & & & & \vdots \\ \vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & 0 & -1 & 2 & -1 & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 & h \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_N \end{pmatrix} = \begin{pmatrix} 0 \\ 2h \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 2h + \\ D \end{pmatrix}$$

14.5 Modification of the element matrix and vector

The modifications of the global linear system can alternatively be done element matrix and vector. (The assembled system will get the value n main diagonal if n elements contribute to the same unknown, but the will also appear on the right-hand side and hence cancel out.)

We have, in the present computational example, the element mat vector (177). The modifications are needed in cells where one of the of freedom is known. Here, this means the first and last cell. We comp element matrix and vector as there are no Dirichlet conditions. The bo term $[u'v]_0^L$ is simply forgotten at nodes that have Dirichlet conditions the modification of the element vector will anyway erase the contributi the boundary term. In the first cell, local degree of freedom number 0 is and the modification becomes

$$\tilde{A}^{(0)} = A = \frac{1}{h}\begin{pmatrix} h & 0 \\ -1 & 1 \end{pmatrix}, \quad \tilde{b}^{(0)} = \begin{pmatrix} 0 \\ h \end{pmatrix}.$$

In the last cell we set

$$\tilde{A}^{(N_e)} = A = \frac{1}{h}\begin{pmatrix} 1 & -1 \\ 0 & h \end{pmatrix}, \quad \tilde{b}^{(N_e)} = \begin{pmatrix} h \\ D \end{pmatrix}.$$

We can also perform the symmetric modification. This operation affe the last cell with a nonzero Dirichlet condition. The algorithm is the for the global linear system, resulting in

$$\tilde{A}^{(N-1)} = A = \frac{1}{h}\begin{pmatrix} h & 0 \\ 0 & 1 \end{pmatrix}, \quad \tilde{b}^{(N-1)} = \begin{pmatrix} h + D/h \\ D \end{pmatrix}.$$

The reader is encouraged to assemble the element matrices and vectors ar that the result coincides with the system (185).

.5 Boundary conditions: specified derivative

uppose our model problem $-u''(x) = f(x)$ features the boundary conditions $'(0) = C$ and $u(L) = D$. As already indicated in Section 12, the former ondition can be incorporated through the boundary term that arises from ıtegration by parts. This details of this method will now be illustrated in the ontext of finite element basis functions.

5.1 The variational formulation

tarting with the Galerkin method,

$$\int_0^L (u''(x) + f(x))\psi_i(x)\,\mathrm{d}x = 0, \quad i \in \mathcal{I}_s,$$

ıtegrating $u''\psi_i$ by parts results in

$$\int_0^L u'(x)'\psi_i'(x)\,\mathrm{d}x - (u'(L)\psi_i(L) - u'(0)\psi_i(0)) = \int_0^L f(x)\psi_i(x)\,\mathrm{d}x, \quad i \in \mathcal{I}_s\,.$$

The first boundary term, $u'(L)\psi_i(L)$, vanishes because $u(L) = D$. There are wo arguments for this result, explained in detail below. The second boundary erm, $u'(0)\psi_i(0)$, can be used to implement the condition $u'(0) = C$, provided $_i(0) \neq 0$ for some i (but with finite elements we fortunately have $\psi_0(0) = 1$). he variational form of the differential equation then becomes

$$\int_0^L u'(x)\varphi_i'(x)\,\mathrm{d}x + C\varphi_i(0) = \int_0^L f(x)\varphi_i(x)\,\mathrm{d}x, \quad i \in \mathcal{I}_s\,.$$

5.2 Boundary term vanishes because of the test functions

t points where u is known we may require ψ_i to vanish. Here, $u(L) = D$ and ıen $\psi_i(L) = 0$, $i \in \mathcal{I}_s$. Obviously, the boundary term $u'(L)\psi_i(L)$ then vanishes.

The set of basis functions $\{\psi_i\}_{i\in\mathcal{I}_s}$ contains in this case all the finite element asis functions on the mesh, expect the one that is 1 at $x = L$. The basis ınction that is left out is used in a boundary function $B(x)$ instead. With a ft-to-right numbering, $\psi_i = \varphi_i$, $i = 0, \ldots, N_n - 1$, and $B(x) = D\varphi_{N_n}$:

$$u(x) = D\varphi_{N_n}(x) + \sum_{j=0}^{N=N_n-1} c_j\varphi_j(x)\,.$$

Inserting this expansion for u in the variational form (15.1) leads to the linear ystem

$$\sum_{j=0}^{N}\left(\int_0^L \varphi_i'(x)\varphi_j'(x)\,\mathrm{d}x\right)c_j = \int_0^L \left(f(x)\varphi_i(x) - D\varphi_{N_n}'(x)\varphi_i(x)\right)\,\mathrm{d}x - C\varphi_i(0), \tag{189}$$

for $i = 0, \ldots, N = N_n - 1$.

15.3 Boundary term vanishes because of linear s modifications

We may, as an alternative to the approach in the previous section, use $\{\psi_i\}_{i\in\mathcal{I}_s}$ which contains all the finite element functions on the mesh: $i = 0, \ldots, N_n = N$. In this case, $u'(L)\psi_i(L) = u'(L)\varphi_i(L) \neq 0$ fo corresponding to the boundary node at $x = L$ (where $\varphi_i = 1$). The nu this node is $i = N_n = N$ if a left-to-right numbering of nodes is utilized

However, even though $u'(L)\varphi_N(L) \neq 0$, we do not need to compute tl For $i < N$ we realize that $\varphi_i(L) = 0$. The only nonzero contributior right-hand side from the affects b_N ($i = N$). Without a boundary func must implement the condition $u(L) = D$ by the equivalent statement and modify the linear system accordingly. This modification will earse row and replace b_N by another value. Any attempt to compute the bc term $u'(L)\varphi_N(L)$ and store it in b_N will be lost. Therefore, we can safel about boundary terms corresponding to Dirichlet boundary conditions al we use the methods from Section 14.3 or Section 14.4.

The expansion for u reads

$$u(x) = \sum_{j\in\mathcal{I}_s} c_j\varphi_j(x), \quad B(x) = D\varphi_N(x),$$

with $N = N_n$. Insertion in the variational form (15.1) leads to the linear

$$\sum_{j\in\mathcal{I}_s}\left(\int_0^L \varphi_i'(x)\varphi_j'(x)\,\mathrm{d}x\right)c_j = \int_0^L (f(x)\varphi_i(x))\,\mathrm{d}x - C\varphi_i(0), \quad i \in \mathcal{I}_s$$

After having computed the system, we replace the last row by $c_N = L$ straightforwardly as in Section refreffem:deq:1D:fem:essBC:Bfunc:modsys symmetric fashion as in Section refreffem:deq:1D:fem:essBC:Bfunc:modsy These modifications can also be performed in the element matrix and ve the right-most cell.

15.4 Direct computation of the global linear system

We now turn to actual computations with P1 finite elements. The foc how the linear system and the element matrices and vectors are modifie condition $u'(0) = C$.

Consider first the approach where Dirichlet conditions are incorporat $B(x)$ function and the known degree of freedom C_{N_n} is left out from tl system (see Section 15.2). The relevant formula for the linear system by (189). There are three differences compared to the extensively co case where $u(0) = 0$ in Sections 13.2 and 13.4. First, because we do not

Dirichlet condition at the left boundary, we need to extend the linear system (172) with an equation associated with the node $x_0 = 0$. According to Section 14.3, this extension consists of including $A_{0,0} = 1/h$, $A_{0,1} = -1/h$, and $b_0 = h$. For $i > 0$ we have $A_{i,i} = 2/h$, $A_{i-1,i} = A_{i,i+1} = -1/h$. Second, we need to include the extra term $-C\varphi_i(0)$ on the right-hand side. Since all $\varphi_i(0) = 0$ for $i = 1, \ldots, N$, this term reduces to $-C\varphi_0(0) = -C$ and affects only the first equation ($i = 0$). We simply add $-C$ to b_0 such that $b_0 = h - C$. Third, the boundary term $-\int_0^L D\varphi_{N_n}(x)\varphi_i \,\mathrm{d}x$ must be computed. Since $i = 0, \ldots, N = N_n - 1$, this integral can only get a nonzero contribution with $i = N_n - 1$ over the last cell. The result becomes $-Dh/6$. The resulting linear system can be summarized in the form

$$\begin{pmatrix} 1 & -1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ -1 & 2 & -1 & \ddots & & & & & \vdots \\ 0 & -1 & 2 & -1 & \ddots & & & & \vdots \\ \vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & 0 & -1 & 2 & -1 & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_N \end{pmatrix} = \begin{pmatrix} h - C \\ 2h \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 2h - Dh/6 \end{pmatrix}. \tag{191}$$

Next we consider the technique where we modify the linear system to incorporate Dirichlet conditions (see Section 15.3). Now $N = N_n$. The two differences from the case above is that the $-\int_0^L D\varphi_{N_n}\varphi_i \,\mathrm{d}x$ term is left out of the right-hand side and an extra last row associated with the node $x_{N_n} = L$ where the Dirichlet condition applies is appended to the system. This last row is anyway replaced by the condition $C_N = D$ or this condition can be incorporated in a symmetric fashion. Using the simplest, former approach gives

$$\frac{1}{h}\begin{pmatrix} 1 & -1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ -1 & 2 & -1 & \ddots & & & & & \vdots \\ 0 & -1 & 2 & -1 & \ddots & & & & \vdots \\ \vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & 0 & -1 & 2 & -1 & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & & \ddots & -1 & 2 & -1 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_N \end{pmatrix} = \begin{pmatrix} h - \\ 2 \\ \\ \\ \\ \\ \\ 2 \\ l \end{pmatrix}$$

15.5 Cellwise computations

Now we compute with one element at a time, working in the reference coo system $X \in [-1, 1]$. We need to see how the $u'(0) = C$ condition the element matrix and vector. The extra term $-C\varphi_i(0)$ in the var formulation only affects the element vector in the first cell. On the refere $-C\varphi_i(0)$ is transformed to $-C\tilde{\varphi}_r(-1)$, where r counts local degrees of f We have $\tilde{\varphi}_0(-1) = 1$ and $\tilde{\varphi}_1(-1) = 0$ so we are left with the contr $-C\tilde{\varphi}_0(-1) = -C$ to $\tilde{b}_0^{(0)}$:

$$\tilde{A}^{(0)} = A = \frac{1}{h}\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}, \quad \tilde{b}^{(0)} = \begin{pmatrix} h - C \\ h \end{pmatrix}.$$

No other element matrices or vectors are affected by the $-C\varphi_i(0)$ bc term.

There are two alternative ways of incorporating the Dirichlet co Following Section 15.2, we get a 1×1 element matrix in the last cell element vector with an extra term containing D:

$$\tilde{A}^{(e)} = \frac{1}{h}\begin{pmatrix} 1 \end{pmatrix}, \quad \tilde{b}^{(e)} = h\begin{pmatrix} 1 - D/6 \end{pmatrix},$$

Alternatively, we include the degree of freedom at the node with u s The element matrix and vector must then be modified to constrain the value at local node $r = 1$:

$$\tilde{A}^{(N_e)} = A = \frac{1}{h}\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad \tilde{b}^{(N_e)} = \begin{pmatrix} h \\ D \end{pmatrix}.$$

.6 Implementation

is tempting to create a program with symbolic calculations to perform all the eps in the computational machinery, both for automating the work and for ocumenting the complete algorithms. As we have seen, there are quite many etails involved with finite element computations and incorporation of boundary onditions. An implementation will also act as a structured summary of all these etails.

6.1 Global basis functions

/e first consider implementations when ψ_i are global functions are hence different om zero on most of $\Omega = [0, L]$ so all integrals need integration over the entire omain. Since the expressions for the entries in the linear system depend on the ifferential equation problem being solved, the user must supply the necessary ormulas via Python functions. The implementations here attempt to perform ymbolic calculations, but fall back on numerical computations if the symbolic nes fail.

The user must prepare a function `integrand_lhs(psi, i, j)` for returning ne integrand of the integral that contributes to matrix entry (i, j). The `psi` ariable is a Python dictionary holding the basis functions and their derivatives n symbolic form. More precisely, `psi[q]` is a list of

$$\{\frac{d^q\psi_0}{dx^q}, \ldots, \frac{d^q\psi_N}{dx^q}\}.$$

imilarly, `integrand_rhs(psi, i)` returns the integrand for entry number i in ne right-hand side vector.

Since we also have contributions to the right-hand side vector, and potentially lso the matrix, from boundary terms without any integral, we introduce two dditional functions, `boundary_lhs(psi, i, j)` and `boundary_rhs(psi, i)` or returning terms in the variational formulation that are not to be integrated ver the domain Ω. Examples shown later will explain in more detail how these ser-supplied function may look like.

The linear system can be computed and solved symbolically by the following unction:

```
import sympy as sp

def solve(integrand_lhs, integrand_rhs, psi, Omega,
          boundary_lhs=None, boundary_rhs=None):
    N = len(psi[0]) - 1
    A = sp.zeros((N+1, N+1))
    b = sp.zeros((N+1, 1))
    x = sp.Symbol('x')
    for i in range(N+1):
        for j in range(i, N+1):
            integrand = integrand_lhs(psi, i, j)
            I = sp.integrate(integrand, (x, Omega[0], Omega[1]))
            if boundary_lhs is not None:
```



```
                I += boundary_lhs(psi, i, j)
            A[i,j] = A[j,i] = I   # assume symmetry
        integrand = integrand_rhs(psi, i)
        I = sp.integrate(integrand, (x, Omega[0], Omega[1]))
        if boundary_rhs is not None:
            I += boundary_rhs(psi, i)
        b[i,0] = I
    c = A.LUsolve(b)
    u = sum(c[i,0]*psi[0][i] for i in range(len(psi[0])))
    return u
```

Not surprisingly, symbolic solution of differential equations, discretiz Galerkin or least squares method with global basis functions, is of limited beyond the simplest problems, because symbolic integration might be ve consuming or impossible, not only in `sympy` but also in WolframAlpha[22] applies the perhaps most powerful symbolic integration software availabl Mathematica). Numerical integration as an option is therefore desirabl

The extended `solve` function below tries to combine symbolic and nu integration. The latter can be enforced by the user, or it can be invoked non-successful symbolic integration (being detected by an `Integral` o the result of the integration in `sympy`). Note that for a numerical inte symbolic expressions must be converted to Python functions (using `lam` and the expressions cannot contain other symbols than `x`. The real `solve` in the `varform1D.py`[23] file has error checking and meaningful error mes such cases. The `solve` code below is a condensed version of the real on the purpose of showing how to automate the Galerkin or least squares for solving differential equations in 1D with global basis functions:

```
def solve(integrand_lhs, integrand_rhs, psi, Omega,
          boundary_lhs=None, boundary_rhs=None, symbolic=True):
    N = len(psi[0]) - 1
    A = sp.zeros((N+1, N+1))
    b = sp.zeros((N+1, 1))
    x = sp.Symbol('x')
    for i in range(N+1):
        for j in range(i, N+1):
            integrand = integrand_lhs(psi, i, j)
            if symbolic:
                I = sp.integrate(integrand, (x, Omega[0], Omega[1
                if isinstance(I, sp.Integral):
                    symbolic = False  # force num.int. hereafter
            if not symbolic:
                integrand = sp.lambdify([x], integrand)
                I = sp.mpmath.quad(integrand, [Omega[0], Omega[1]
            if boundary_lhs is not None:
                I += boundary_lhs(psi, i, j)
            A[i,j] = A[j,i] = I
        integrand = integrand_rhs(psi, i)
        if symbolic:
            I = sp.integrate(integrand, (x, Omega[0], Omega[1]))
            if isinstance(I, sp.Integral):
                symbolic = False
```

[22] http://wolframalpha.com
[23] http://tinyurl.com/nm5587k/fem/varform1D.py

```
        if not symbolic:
            integrand = sp.lambdify([x], integrand)
            I = sp.mpmath.quad(integrand, [Omega[0], Omega[1]])
        if boundary_rhs is not None:
            I += boundary_rhs(psi, i)
        b[i,0] = I
    c = A.LUsolve(b)
    u = sum(c[i,0]*psi[0][i] for i in range(len(psi[0])))
    return u
```

6.2 Example: constant right-hand side

o demonstrate the code above, we address

$$-u''(x) = b, \quad x \in \Omega = [0,1], \quad u(0) = 1,\ u(1) = 0,$$

ith b as a (symbolic) constant. A possible basis for the space V is $\psi_i(x) = {}^{i+1}(1-x)$, $i \in \mathcal{I}_s$. Note that $\psi_i(0) = \psi_i(1) = 0$ as required by the Dirichlet onditions. We need a $B(x)$ function to take care of the known boundary values f u. Any function $B(x) = 1 - x^p$, $p \in \mathbb{R}$, is a candidate, and one arbitrary hoice from this family is $B(x) = 1 - x^3$. The unknown function is then written s

$$u(x) = B(x) + \sum_{j \in \mathcal{I}_s} c_j \psi_j(x)\,.$$

Let us use the Galerkin method to derive the variational formulation. Multiplying the differential equation by v and integrate by parts yield

$$\int_0^1 u'v'\,\mathrm{d}x = \int_0^1 fv\,\mathrm{d}x \quad \forall v \in V,$$

nd with $u = B + \sum_j c_j\psi_j$ we get the linear system

$$\sum_{j \in \mathcal{I}_s} \left(\int_0^1 \psi_i'\psi_j'\,\mathrm{d}x \right) c_j = \int_0^1 (f - B')\psi_i\,\mathrm{d}x, \quad i \in \mathcal{I}_s\,. \tag{196}$$

The application can be coded as follows in `sympy`:

```
x, b = sp.symbols('x b')
f = b
B = 1 - x**3
dBdx = sp.diff(B, x)

# Compute basis functions and their derivatives
N = 3
psi = {0: [x**(i+1)*(1-x) for i in range(N+1)]}
psi[1] = [sp.diff(psi_i, x) for psi_i in psi[0]]

def integrand_lhs(psi, i, j):
    return psi[1][i]*psi[1][j]
```



```
def integrand_rhs(psi, i):
    return f*psi[0][i] - dBdx*psi[1][i]

Omega = [0, 1]

u_bar = solve(integrand_lhs, integrand_rhs, psi, Omega,
              verbose=True, symbolic=True)
u = B + u_bar
print 'solution u:', sp.simplify(sp.expand(u))
```

The printout of `u` reads `-b*x**2/2 + b*x/2 - x + 1`. Note that expa and then simplifying is in the present case necessary to get a compa expression with `sympy`. A non-expanded `u` might be preferable in othe this depends on the problem in question.

The exact solution $u_{\mathrm{e}}(x)$ can be derived by some `sympy` code that follows the examples in Section 11.2. The idea is to integrate $-u'' = b$ tw determine the integration constants from the boundary conditions:

```
C1, C2 = sp.symbols('C1 C2')    # integration constants
f1 = sp.integrate(f, x) + C1
f2 = sp.integrate(f1, x) + C2
# Find C1 and C2 from the boundary conditions u(0)=0, u(1)=1
s = sp.solve([u_e.subs(x,0) - 1, u_e.subs(x,1) - 0], [C1, C2])
# Form the exact solution
u_e = -f2 + s[C1]*x + s[C2]
print 'analytical solution:', u_e
print 'error:', sp.simplify(sp.expand(u - u_e))
```

The last line prints `0`, which is not surprising when $u_{\mathrm{e}}(x)$ is a parabola approximate u contains polynomials up to degree 4. It suffices to have i.e., polynomials of degree 2, to recover the exact solution.

We can play around with the code and test that with $f \sim x^p$, the so a polynomial of degree $p+2$, and $N = p+1$ guarantees that the appr solution is exact.

Although the symbolic code is capable of integrating many choices the symbolic expressions for u quickly become lengthy and non-info so numerical integration in the code, and hence numerical answers, h greatest application potential.

16.3 Finite elements

Implementation of the finite element algorithms for differential equat lows closely the algorithm for approximation of functions. The new ad ingredients are

1. other types of integrands (as implied by the variational formulati
2. additional boundary terms in the variational formulation for N boundary conditions
3. modification of element matrices and vectors due to Dirichlet bc conditions

oint 1 and 2 can be taken care of by letting the user supply functions defining ıe integrands and boundary terms on the left- and right-hand side of the quation system:

```
integrand_lhs(phi, r, s, x)
boundary_lhs(phi, r, s, x)
integrand_rhs(phi, r, x)
boundary_rhs(phi, r, x)
```

ere, `phi` is a dictionary where `phi[q]` holds a list of the derivatives of order of the basis functions at the an evaluation point; `r` and `s` are indices for the orresponding entries in the element matrix and vector, and `x` is the global oordinate value corresponding to the current evaluation point.

Given a mesh represented by `vertices`, `cells`, and `dof_map` as explained be-ore, we can write a pseudo Python code to list all the steps in the computational lgorithm for finite element solution of a differential equation.

```
<Declare global matrix and rhs: A, b>

for e in range(len(cells)):

    # Compute element matrix and vector
    n = len(dof_map[e])  # no of dofs in this element
    h = vertices[cells[e][1]] - vertices[cells[e][1]]
    <Declare element matrix and vector: A_e, b_e>

    # Integrate over the reference cell
    points, weights = <numerical integration rule>
    for X, w in zip(points, weights):
        phi = <basis functions and derivatives at X>
        detJ = h/2
        x = <affine mapping from X>
        for r in range(n):
            for s in range(n):
                A_e[r,s] += integrand_lhs(phi, r, s, x)*detJ*w
            b_e[r] += integrand_rhs(phi, r, x)*detJ*w

    # Add boundary terms
    for r in range(n):
        for s in range(n):
            A_e[r,s] += boundary_lhs(phi, r, s, x)*detJ*w
        b_e[r] += boundary_rhs(phi, r, x)*detJ*w

    # Incorporate essential boundary conditions
    for r in range(n):
        global_dof = dof_map[e][r]
        if global_dof in essbc_dofs:
            # dof r is subject to an essential condition
            value = essbc_docs[global_dof]
            # Symmetric modification
            b_e -= value*A_e[:,r]
            A_e[r,:] = 0
            A_e[:,r] = 0
            A_e[r,r] = 1
            b_e[r] = value

    # Assemble
```



```
    for r in range(n):
        for s in range(n):
            A[dof_map[e][r], dof_map[e][r]] += A_e[r,s]
        b[dof_map[e][r] += b_e[r]

<solve linear system>
```

17 Variational formulations in 2D and 3D

The major difference between deriving variational formulations in 2D compared to 1D is the rule for integrating by parts. A typical second-orc in a PDE may be written in dimension-independent notation as

$$\nabla^2 u \quad \text{or} \quad \nabla\cdot(a(\boldsymbol{x})\nabla u)\ .$$

The explicit forms in a 2D problem become

$$\nabla^2 u = \nabla\cdot\nabla u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2},$$

and

$$\nabla\cdot(a(\boldsymbol{x})\nabla u) = \frac{\partial}{\partial x}\left(a(x,y)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(a(x,y)\frac{\partial u}{\partial y}\right)\ .$$

We shall continue with the latter operator as the form arises from just $a = 1$.

The general rule for integrating by parts is often referred to as Gree identity[24]:

$$-\int_\Omega \nabla\cdot(a(\boldsymbol{x})\nabla u)v\,\mathrm{d}x = \int_\Omega a(\boldsymbol{x})\nabla u\cdot\nabla v\,\mathrm{d}x - \int_{\partial\Omega} a\frac{\partial u}{\partial n}v\,\mathrm{d}s,$$

where $\partial\Omega$ is the boundary of Ω and $\partial u/\partial n = \boldsymbol{n}\cdot\nabla u$ is the derivative of outward normal direction, $\boldsymbol{n}$ being an outward unit normal to $\partial\Omega$. The i $\int_\Omega()\,\mathrm{d}x$ are area integrals in 2D and volume integrals in 3D, while $\int_{\partial\Omega}($ line integral in 2D and a surface integral in 3D.

Let us divide the boundary into two parts:

- $\partial\Omega_N$, where we have Neumann conditions $-a\frac{\partial u}{\partial n} = g$, and
- $\partial\Omega_D$, where we have Dirichlet conditions $u = u_0$.

The test functions v are required to vanish on $\partial\Omega_D$.

[24] http://en.wikipedia.org/wiki/Green's_identities

Example. Here is a quite general, stationary, linear PDE arising in many problems:

$$\boldsymbol{v}\cdot\nabla u + \alpha u = \nabla\cdot(a\nabla u) + f, \quad \boldsymbol{x}\in\Omega, \tag{198}$$

$$u = u_0, \quad \boldsymbol{x}\in\partial\Omega_D, \tag{199}$$

$$-a\frac{\partial u}{\partial n} = g, \quad \boldsymbol{x}\in\partial\Omega_N\,. \tag{200}$$

The vector field $\boldsymbol{v}$ and the scalar functions a, α, f, u_0, and g may vary with the spatial coordinate $\boldsymbol{x}$ and must be known.

Such a second-order PDE needs exactly one boundary condition at each point of the boundary, so $\partial\Omega_N\cup\partial\Omega_D$ must be the complete boundary $\partial\Omega$.

Assume that the boundary function $u_0(\boldsymbol{x})$ is defined for all $\boldsymbol{x}\in\Omega$. The unknown function can then be expanded as

$$u = B + \sum_{j\in\mathcal{I}_s} c_j\psi_j, \quad B = u_0\,.$$

The variational formula is obtained from Galerkin's method, which technically implies multiplying the PDE by a test function v and integrating over Ω:

$$\int_\Omega (\boldsymbol{v}\cdot\nabla u + \alpha u)v\,\mathrm{d}x = \int_\Omega \nabla\cdot(a\nabla u)\,\mathrm{d}x + \int_\Omega fv\,\mathrm{d}x\,.$$

The second-order term is integrated by parts, according to

$$\int_\Omega \nabla\cdot(a\nabla u)\,v\,\mathrm{d}x = -\int_\Omega a\nabla u\cdot\nabla v\,\mathrm{d}x + \int_{\partial\Omega} a\frac{\partial u}{\partial n}v\,\mathrm{d}s\,.$$

The variational form now reads

$$\int_\Omega (\boldsymbol{v}\cdot\nabla u + \alpha u)v\,\mathrm{d}x = -\int_\Omega a\nabla u\cdot\nabla v\,\mathrm{d}x + \int_{\partial\Omega} a\frac{\partial u}{\partial n}v\,\mathrm{d}s + \int_\Omega fv\,\mathrm{d}x\,.$$

The boundary term can be developed further by noticing that $v\neq 0$ only on Ω_N,

$$\int_{\partial\Omega} a\frac{\partial u}{\partial n}v\,\mathrm{d}s = \int_{\partial\Omega_N} a\frac{\partial u}{\partial n}v\,\mathrm{d}s,$$

and that on $\partial\Omega_N$, we have the condition $a\frac{\partial u}{\partial n} = -g$, so the term becomes

$$-\int_{\partial\Omega_N} gv\,\mathrm{d}s\,.$$

The variational form is then

$$\int_\Omega (\boldsymbol{v}\cdot\nabla u + \alpha u)v\,\mathrm{d}x = -\int_\Omega a\nabla u\cdot\nabla v\,\mathrm{d}x - \int_{\partial\Omega_N} gv\,\mathrm{d}s + \int_\Omega fv\,\mathrm{d}x\,.$$

Instead of using the integral signs we may use the inner product no

$$(\boldsymbol{v}\cdot\nabla u, v) + (\alpha u, v) = -(a\nabla u, \nabla v) - (g, v)_N + (f, v)\,.$$

The subscript $_N$ in $(g, v)_N$ is a notation for a line or surface integral ov

Inserting the u expansion results in

$$\sum_{j\in\mathcal{I}_s}((\boldsymbol{v}\cdot\nabla\psi_j, \psi_i) + (\alpha\psi_j, \psi_i) + (a\nabla\psi_j, \nabla\psi_i))c_j =$$

$$(g, \psi_i)_N + (f, \psi_i) - (\boldsymbol{v}\cdot\nabla u_0, \psi_i) + (\alpha u_0, \psi_i) + (a\nabla u_0$$

This is a linear system with matrix entries

$$A_{i,j} = (\boldsymbol{v}\cdot\nabla\psi_j, \psi_i) + (\alpha\psi_j, \psi_i) + (a\nabla\psi_j, \nabla\psi_i)$$

and right-hand side entries

$$b_i = (g, \psi_i)_N + (f, \psi_i) - (\boldsymbol{v}\cdot\nabla u_0, \psi_i) + (\alpha u_0, \psi_i) + (a\nabla u_0, \nabla\psi_i)$$

for $i, j\in\mathcal{I}_s$.

In the finite element method, we usually express u_0 in terms of basis fu and restrict i and j to run over the degrees of freedom that are not presc Dirichlet conditions. However, we can also keep all the c_j, $j\in\mathcal{I}_s$, as un drop the u_0 in the expansion for u, and incorporate all the known c_j v the linear system. This has been explained in detail in the 1D case.

17.1 Transformation to a reference cell in 2D and 3

We consider an integral of the type

$$\int_{\Omega^{(e)}} a(\boldsymbol{x})\nabla\varphi_i\cdot\nabla\varphi_j\,\mathrm{d}x,$$

where the φ_i functions are finite element basis functions in 2D or 3D, de the physical domain. Suppose we want to calculate this integral over a r cell, denoted by $\tilde{\Omega}^r$, in a coordinate system with coordinates $\boldsymbol{X} = (X_0, X$ or $\boldsymbol{X} = (X_0, X_1, X_2)$ (3D). The mapping between a point $\boldsymbol{X}$ in the re coordinate system and the corresponding point $\boldsymbol{x}$ in the physical coo system is given by a vector relation $\boldsymbol{x}(\boldsymbol{X})$. The corresponding Jacobia this mapping has entries

$$J_{i,j} = \frac{\partial x_j}{\partial X_i}\,.$$

The change of variables requires $\mathrm{d}x$ to be replaced by $\det J\,\mathrm{d}X$. The tives in the ∇ operator in the variational form are with respect to $\boldsymbol{x}$ we may denote by $\nabla_{\boldsymbol{x}}$. The $\varphi_i(\boldsymbol{x})$ functions in the integral are repl local basis functions $\tilde{\varphi}_r(\boldsymbol{X})$ so the integral features $\nabla_{\boldsymbol{x}}\tilde{\varphi}_r(\boldsymbol{X})$. We read

$\nabla_{\boldsymbol{X}}\tilde{\varphi}_r(\boldsymbol{X})$ from formulas for the basis functions in the reference cell, but the esired quantity $\nabla_{\boldsymbol{x}}\tilde{\varphi}_r(\boldsymbol{X})$ requires some efforts to compute. All the details are rovided below.

Let $i = q(e, r)$ and consider two space dimensions. By the chain rule,

$$\frac{\partial\tilde{\varphi}_r}{\partial X} = \frac{\partial\varphi_i}{\partial X} = \frac{\partial\varphi_i}{\partial x}\frac{\partial x}{\partial X} + \frac{\partial\varphi_i}{\partial y}\frac{\partial y}{\partial X},$$

nd

$$\frac{\partial\tilde{\varphi}_r}{\partial Y} = \frac{\partial\varphi_i}{\partial Y} = \frac{\partial\varphi_i}{\partial x}\frac{\partial x}{\partial Y} + \frac{\partial\varphi_i}{\partial y}\frac{\partial y}{\partial Y}.$$

/e can write these two equations as a vector equation

$$\begin{bmatrix} \frac{\partial\tilde{\varphi}_r}{\partial X} \\ \frac{\partial\tilde{\varphi}_r}{\partial Y} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial X} & \frac{\partial y}{\partial X} \\ \frac{\partial x}{\partial Y} & \frac{\partial y}{\partial Y} \end{bmatrix} \begin{bmatrix} \frac{\partial\varphi_i}{\partial x} \\ \frac{\partial\varphi_i}{\partial y} \end{bmatrix}$$

lentifying

$$\nabla_{\boldsymbol{X}}\tilde{\varphi}_r = \begin{bmatrix} \frac{\partial\tilde{\varphi}_r}{\partial X} \\ \frac{\partial\tilde{\varphi}_r}{\partial Y} \end{bmatrix}, \quad J = \begin{bmatrix} \frac{\partial x}{\partial X} & \frac{\partial y}{\partial X} \\ \frac{\partial x}{\partial Y} & \frac{\partial y}{\partial Y} \end{bmatrix}, \quad \nabla_{\boldsymbol{x}}\varphi_r = \begin{bmatrix} \frac{\partial\varphi_i}{\partial x} \\ \frac{\partial\varphi_i}{\partial y} \end{bmatrix},$$

e have the relation

$$\nabla_{\boldsymbol{X}}\tilde{\varphi}_r = J\cdot\nabla_{\boldsymbol{x}}\varphi_i,$$

hich we can solve with respect to $\nabla_{\boldsymbol{x}}\varphi_i$:

$$\nabla_{\boldsymbol{x}}\varphi_i = J^{-1}\cdot\nabla_{\boldsymbol{X}}\tilde{\varphi}_r\,. \tag{202}$$

n the reference cell, $\varphi_i(\boldsymbol{x}) = \tilde{\varphi}_r(\boldsymbol{X})$, so

$$\nabla_{\boldsymbol{x}}\tilde{\varphi}_r(\boldsymbol{X}) = J^{-1}(\boldsymbol{X})\cdot\nabla_{\boldsymbol{X}}\tilde{\varphi}_r(\boldsymbol{X})\,. \tag{203}$$

This means that we have the following transformation of the integral in the hysical domain to its counterpart over the reference cell:

$$\int_\Omega^{(e)} a(\boldsymbol{x})\nabla_{\boldsymbol{x}}\varphi_i\cdot\nabla_{\boldsymbol{x}}\varphi_j\,\mathrm{d}x \int_{\tilde{\Omega}^r} a(\boldsymbol{x}(\boldsymbol{X}))(J^{-1}\cdot\nabla_{\boldsymbol{X}}\tilde{\varphi}_r)\cdot(J^{-1}\cdot\nabla\tilde{\varphi}_s)\det J\,\mathrm{d}X \tag{204}$$

7.2 Numerical integration

ntegrals are normally computed by numerical integration rules. For multi-imensional cells, various families of rules exist. All of them are similar to hat is shown in 1D: $\int f\,\mathrm{d}x \approx \sum_j w_i f(\boldsymbol{x}_j)$, where w_j are weights and $\boldsymbol{x}_j$ are orresponding points.

The file `numint.py`[25] contains the functions `quadrature_for_triangles(n)` nd `quadrature_for_tetrahedra(n)`, which returns lists of points and weights

[25] `http://tinyurl.com/nm5587k/fem/numint.py`

corresponding to integration rules with `n` points over the reference with vertices $(0,0)$, $(1,0)$, $(0,1)$, and the reference tetrahedron with $(0,0,0)$, $(1,0,0)$, $(0,1,0)$, $(0,0,1)$, respectively. For example, the first tv for integration over a triangle have 1 and 3 points:

```
>>> import numint
>>> x, w = numint.quadrature_for_triangles(num_points=1)
>>> x
[(0.3333333333333333, 0.3333333333333333)]
>>> w
[0.5]
>>> x, w = numint.quadrature_for_triangles(num_points=3)
>>> x
[(0.16666666666666666, 0.16666666666666666),
 (0.66666666666666666, 0.16666666666666666),
 (0.16666666666666666, 0.66666666666666666)]
>>> w
[0.16666666666666666, 0.16666666666666666, 0.16666666666666666]
```

Rules with 1, 3, 4, and 7 points over the triangle will exactly integrate poly of degree 1, 2, 3, and 4, respectively. In 3D, rules with 1, 4, 5, and 1 over the tetrahedron will exactly integrate polynomials of degree 1, 2, 3 respectively.

17.3 Convenient formulas for P1 elements in 2D

We shall now provide some formulas for piecewise linear φ_i functions a integrals *in the physical coordinate system*. These formulas make it conve compute with P1 elements without the need to work in the reference coo system and deal with mappings and Jacobians. A lot of computatio algorithmic details are hidden by this approach.

Let $\Omega^{(e)}$ be cell number e, and let the three vertices have globa numbers I, J, and K. The corresponding coordinates are (x_I, y_I), $(x_J, y$ (x_K, y_K). The basis function φ_I over $\Omega^{(e)}$ have the explicit formula

$$\varphi_I(x,y) = \frac{1}{2}\Delta\left(\alpha_I + \beta_I x + \gamma_I y\right),$$

where

$$\begin{aligned}
\alpha_I &= x_J y_K - x_K y_J,\\
\beta_I &= y_J - y_K,\\
\gamma_I &= x_K - x_J,\\
2\Delta &= \det\begin{pmatrix} 1 & x_I & y_I \\ 1 & x_J & y_J \\ 1 & x_K & y_K \end{pmatrix}.
\end{aligned}$$

The quantity Δ is the area of the cell.

The following formula is often convenient when computing element matrices nd vectors:

$$\int_{\Omega^{(e)}} \varphi_I^p \varphi_J^q \varphi_K^r dxdy = \frac{p!q!r!}{(p+q+r+2)!} 2\Delta\,. \tag{210}$$

Note that the q in this formula is not to be mixed with the $q(e,r)$ mapping of egrees of freedom.)

As an example, the element matrix entry $\int_{\Omega^{(e)}} \varphi_I \varphi_J \,\mathrm{d}x$ can be computed y setting $p = q = 1$ and $r = 0$, when $I \neq J$, yielding $\Delta/12$, and $p = 2$ and $= r = 0$, when $I = J$, resulting in $\Delta/6$. We collect these numbers in a local ement matrix:

$$\frac{\Delta}{12}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

The common element matrix entry $\int_{\Omega^{(e)}} \nabla\varphi_I \cdot \nabla\varphi_J \,\mathrm{d}x$, arising from a Laplace erm $\nabla^2 u$, can also easily be computed by the formulas above. We have

$$\nabla\varphi_I \cdot \nabla\varphi_J = \frac{\Delta^2}{4}(\beta_I\beta_J + \gamma_I\gamma_J) = \text{const},$$

o that the element matrix entry becomes $\frac{1}{4}\Delta^3(\beta_I\beta_J + \gamma_I\gamma_J)$.

From an implementational point of view, one will work with local vertex umbers $r = 0, 1, 2$, parameterize the coefficients in the basis functions by r, and ook up vertex coordinates through $q(e,r)$.

Similar formulas exist for integration of P1 elements in 3D.

.8 Summary

- When approximating f by $u = \sum_j c_j\varphi_j$, the least squares method and the Galerkin/projection method give the same result. The interpolation/collocation method is simpler and yields different (mostly inferior) results.
- Fourier series expansion can be viewed as a least squares or Galerkin approximation procedure with sine and cosine functions.
- Basis functions should optimally be orthogonal or almost orthogonal, because this gives little round-off errors when solving the linear system, and the coefficient matrix becomes diagonal or sparse.
- Finite element basis functions are *piecewise* polynomials, normally with discontinuous derivatives at the cell boundaries. The basis functions overlap very little, leading to stable numerics and sparse matrices.
- To use the finite element method for differential equations, we use the Galerkin method or the method of weighted residuals to arrive at a variational form. Technically, the differential equation is multiplied by a test

function and integrated over the domain. Second-order derivat integrated by parts to allow for typical finite element basis functio have discontinuous derivatives.

- The least squares method is not much used for finite element sol differential equations of second order, because it then involves secor derivatives which cause trouble for basis functions with discon derivatives.
- We have worked with two common finite element terminologies an ated data structures (both are much used, especially the first on the other is more general):
  1. *elements*, *nodes*, and *mapping between local and global node n*
  2. an extended element concept consisting of *cell*, *vertices*, *de freedom*, *local basis functions*, *geometry mapping*, and *mapping local and global degrees of freedom*
- The meaning of the word "element" is multi-fold: the geometry of element (also known as a cell), the geometry and its basis function information listed under point 2 above.
- One normally computes integrals in the finite element method by element (cell by cell), either in a local reference coordinate sy directly in the physical domain.
- The advantage of working in the reference coordinate system is t mathematical expressions for the basis functions depend on the type only, not the geometry of that element in the physical doma disadvantage is that a mapping must be used, and derivatives n transformed from reference to physical coordinates.
- Element contributions to the global linear system are collected in an matrix and vector, which must be assembled into the global syste the degree of freedom mapping (`dof_map`) or the node numbering n (`elements`), depending on which terminology that is used.
- Dirichlet conditions, involving prescribed values of u at the bound implemented either via a boundary function that take on the right I values, while the basis functions vanish at such boundaries. In tl element method, one has a general expression for the boundary f but one can also incorporate Dirichlet conditions in the element and vector or in the global matrix system.
- Neumann conditions, involving prescribed values of the deriva flux) of u, are incorporated in boundary terms arising from inte terms with second-order derivatives by part. Forgetting to account boundary terms implies the condition $\partial u/\partial n = 0$ at parts of the b where no Dirichlet condition is set.

.9 Time-dependent problems

he finite element method is normally used for discretization in space. There re two alternative strategies for performing a discretization in time:

- use finite differences for time derivatives to arrive at a recursive set of spatial problems that can be discretized by the finite element method, or
- discretize in space by finite elements first, and then solve the resulting system of ordinary differential equations (ODEs) by some standard method for ODEs.

/e shall exemplify these strategies using a simple diffusion problem

$$\frac{\partial u}{\partial t} = \alpha\nabla^2 u + f(\boldsymbol{x},t), \qquad \boldsymbol{x}\in\Omega, t\in(0,T], \tag{211}$$

$$u(\boldsymbol{x},0) = I(\boldsymbol{x}), \qquad \boldsymbol{x}\in\Omega, \tag{212}$$

$$\frac{\partial u}{\partial n} = 0, \qquad \boldsymbol{x}\in\partial\Omega,\ t\in(0,T]\,. \tag{213}$$

ere, $u(\boldsymbol{x},t)$ is the unknown function, α is a constant, and $f(\boldsymbol{x},t)$ and $I(x)$ re given functions. We have assigned the particular boundary condition (213) o minimize the details on handling boundary conditions in the finite element ıethod.

9.1 Discretization in time by a Forward Euler scheme

'ime discretization. We can apply a finite difference method in time to !11). First we need a mesh in time, here taken as uniform with mesh points $_i = n\Delta t$, $n = 0,1,\ldots,N_t$. A Forward Euler scheme consists of sampling !11) at t_n and approximating the time derivative by a forward difference $D_t^+u]^n \approx (u^{n+1}-u^n)/\Delta t$. This approximation turns (211) into a differential quation that is discrete in time, but still continuous in space. With a finite ifference operator notation we can write the time-discrete problem as

$$[D_t^+ u = \alpha\nabla^2 u + f]^n, \tag{214}$$

or $n = 1,2,\ldots,N_t-1$. Writing this equation out in detail and isolating the nknown u^{n+1} on the left-hand side, demonstrates that the time-discrete problem a recursive set of problems that are continuous in space:

$$u^{n+1} = u^n + \Delta t\left(\alpha\nabla^2 u^n + f(\boldsymbol{x},t_n)\right)\,. \tag{215}$$

iven $u^0 = I$, we can use (215) to compute $u^1,u^2,\ldots,u^{N_t}$.

For absolute clarity in the various stages of the discretizations, we introduce $_e(\boldsymbol{x},t)$ as the exact solution of the space-and time-continuous partial differential quation (211) and $u_e^n(\boldsymbol{x})$ as the time-discrete approximation, arising from the nite difference method in time (214). More precisely, u_e fulfills

$$\frac{\partial u_e}{\partial t} = \alpha\nabla^2 u_e + f(\boldsymbol{x},t),$$

while u_e^{n+1}, with a superscript, is the solution of the time-discrete equa

$$u_e^{n+1} = u_e^n + \Delta t\left(\alpha\nabla^2 u_e^n + f(\boldsymbol{x},t_n)\right)\,.$$

Space discretization. We now introduce a finite element approxim u_e^n and u_e^{n+1} in (217), where the coefficients depend on the time level:

$$u_e^n \approx u^n = \sum_{j=0}^{N} c_j^n\psi_j(\boldsymbol{x}),$$

$$u_e^{n+1} \approx u^{n+1} = \sum_{j=0}^{N} c_j^{n+1}\psi_j(\boldsymbol{x})\,.$$

Note that, as before, N denotes the number of degrees of freedom in th domain. The number of time points is denoted by N_t. We define a s spanned by the basis functions $\{\psi_i\}_{i\in\mathcal{I}_s}$.

19.2 Variational forms

A weighted residual method with weighting functions w_i can now be for We insert (218) and (219) in (217) to obtain the residual

$$R = u^{n+1} - u^n - \Delta t\left(\alpha\nabla^2 u^n + f(\boldsymbol{x},t_n)\right)\,.$$

The weighted residual principle,

$$\int_\Omega Rw\,\mathrm{d}x = 0, \quad \forall w\in W,$$

results in

$$\int_\Omega \left[u^{n+1} - u^n - \Delta t\left(\alpha\nabla^2 u^n + f(\boldsymbol{x},t_n)\right)\right] w\,\mathrm{d}x = 0, \quad \forall w\in W\,.$$

From now on we use the Galerkin method so $W = V$. Isolating the u u^{n+1} on the left-hand side gives

$$\int_\Omega u^{n+1}\psi_i\,\mathrm{d}x = \int_\Omega \left[u^n - \Delta t\left(\alpha\nabla^2 u^n + f(\boldsymbol{x},t_n)\right)\right] v\,\mathrm{d}x, \quad \forall v\in V$$

As usual in spatial finite element problems involving second-order der we apply integration by parts on the term $\int(\nabla^2 u^n)v\,\mathrm{d}x$:

$$\int_\Omega \alpha(\nabla^2 u^n)v\,\mathrm{d}x = -\int_\Omega \alpha\nabla u^n\cdot\nabla v\,\mathrm{d}x + \int_{\partial\Omega}\alpha\frac{\partial u^n}{\partial n}v\,\mathrm{d}x\,.$$

he last term vanishes because we have the Neumann condition $\partial u^n/\partial n = 0$ for ll n. Our discrete problem in space and time then reads

$$\int_\Omega u^{n+1} v \,\mathrm{d}x = \int_\Omega u^n v dx - \Delta t \int_\Omega \alpha \nabla u^n \cdot \nabla v \,\mathrm{d}x + \Delta t \int_\Omega f^n v \,\mathrm{d}x, \quad \forall \boldsymbol{v} \in V\,. \tag{220}$$

his is the variational formulation of our recursive set of spatial problems.

Nonzero Dirichlet boundary conditions.

As in stationary problems, we can introduce a boundary function $B(\boldsymbol{x}, t)$ to take care of nonzero Dirichlet conditions:

$$u_{\mathrm{e}}^n \approx u^n = B(\boldsymbol{x}, t_n) + \sum_{j=0}^N c_j^n \psi_j(\boldsymbol{x}), \tag{221}$$

$$u_{\mathrm{e}}^{n+1} \approx u^{n+1} = B(\boldsymbol{x}, t_{n+1}) + \sum_{j=0}^N c_j^{n+1} \psi_j(\boldsymbol{x})\,. \tag{222}$$

9.3 Simplified notation for the solution at recent time levels

ı a program it is only necessary to store u^{n+1} and u^n at the same time. We ıerefore drop the n index in programs and work with two functions: `u` for u^{n+1}, ıe new unknown, and `u_1` for u^n, the solution at the previous time level. This also convenient in the mathematics to maximize the correspondence with ıe code. From now on u_1 means the discrete unknown at the previous time evel (u^n) and u represents the discrete unknown at the new time level (u^{n+1}). quation (220) with this new naming convention is expressed as

$$\int_\Omega uv dx = \int_\Omega u_1 v dx - \Delta t \int_\Omega \alpha \nabla u_1 \cdot \nabla v \,\mathrm{d}x + \Delta t \int_\Omega f^n v \,\mathrm{d}x\,. \tag{223}$$

his variational form can alternatively be expressed by the inner product notaon:

$$(u, v) = (u_1, v) - \Delta t(\alpha \nabla u_1, \nabla v) + (f^n, v)\,. \tag{224}$$

9.4 Deriving the linear systems

o derive the equations for the new unknown coefficients c_j^{n+1}, now just called $_j$, we insert

$$u = \sum_{j=0}^N c_j \psi_j(\boldsymbol{x}), \quad u_1 = \sum_{j=0}^N c_{1,j} \psi_j(\boldsymbol{x})$$

in (223) or (224), let the equation hold for all $v = \psi$, $i = 0, \ldots,$N, and o terms as matrix-vector products:

$$\sum_{j=0}^N (\psi_i, \psi_j) c_j = \sum_{j=0}^N (\psi_i, \psi_j) c_{1,j} - \Delta t \sum_{j=0}^N (\nabla \psi_i, \alpha \nabla \psi_j) c_{1,j} + (f^n, \psi_i), \quad i =$$

This is a linear system $\sum_j A_{i,j} c_j = b_i$ with

$$A_{i,j} = (\psi_i, \psi_j)$$

and

$$b_i = \sum_{j=0}^N (\psi_i, \psi_j) c_{1,j} - \Delta t \sum_{j=0}^N (\nabla \psi_i, \alpha \nabla \psi_j) c_{1,j} + (f^n, \psi_i)\,.$$

It is instructive and convenient for implementations to write the linear on the form

$$Mc = Mc_1 - \Delta t K c_1 + f,$$

where

$$\begin{aligned} M &= \{M_{i,j}\}, \quad M_{i,j} = (\psi_i, \psi_j), \quad i, j \in \mathcal{I}_s, \\ K &= \{K_{i,j}\}, \quad K_{i,j} = (\nabla \psi_i, \alpha \nabla \psi_j), \quad i, j \in \mathcal{I}_s, \\ f &= \{(f(\boldsymbol{x}, t_n), \psi_i)\}_{i \in \mathcal{I}_s}, \\ c &= \{c_i\}_{i \in \mathcal{I}_s}, \\ c_1 &= \{c_{1,i}\}_{i \in \mathcal{I}_s}\,. \end{aligned}$$

We realize that M is the matrix arising from a term with the zero-th de of u, and called the mass matrix, while K is the matrix arising from a term $\nabla^2 u$. The K matrix is often known as the *stiffness matrix*. (Th mass and stiffness stem from the early days of finite elements when appl to vibrating structures dominated. The mass matrix arises from the ma acceleration term in Newton's second law, while the stiffness matrix aris the elastic forces in that law. The mass and stiffness matrix appear diffusion have slightly different mathematical formulas.)

Remark. The mathematical symbol f has two meanings, either the f $f(\boldsymbol{x}, t)$ in the PDE or the f vector in the linear system to be solved at ea level. The symbol u also has different meanings, basically the unknow PDE or the finite element function representing the unknown at a tir The actual meaning should be evident from the context.

9.5 Computational algorithm

We observe that M and K can be precomputed so that we can avoid computing the matrix entries at every time level. Instead, some matrix-vector multiplications will produce the linear system to be solved. The computational algorithm has the following steps:

1. Compute M and K.
2. Initialize u^0 by interpolation or projection
3. For $n = 1, 2, \ldots, N_t$:
   (a) compute $b = Mc_1 - \Delta t K c_1 + f$
   (b) solve $Mc = b$
   (c) set $c_1 = c$

In case of finite element basis functions, interpolation of the initial condition at the nodes means $c_{1,j} = I(\boldsymbol{x}_j)$. Otherwise one has to solve the linear system $\sum_j \psi_j(x_i)c_j = I(x_i)$, where $\boldsymbol{x}_j$ denotes an interpolation point. Projection (or Galerkin's method) implies solving a linear system with M as coefficient matrix $\sum_j M_{i,j}c_{1,j} = (I, \psi_i)$, $i \in \mathcal{I}_s$.

9.6 Comparing P1 elements with the finite difference method

We can compute the M and K matrices using P1 elements in 1D. A uniform mesh on $[0, L]$ is introduced for this purpose. Since the boundary conditions are solely of Neumann type in this sample problem, we have no restrictions on the basis functions ψ_i and can simply choose $\psi_i = \varphi_i$, $i = 0, \ldots, N = N_n$.

From Section 13.2 or 13.4 we have that the K matrix is the same as we get from the finite difference method: $h[D_xD_xu]_i^n$, while from Section 5.2 we know that M can be interpreted as the finite difference approximation $[u+\frac{1}{6}h^2D_xD_xu]_i^n$ times h). The equation system $Mc = b$ in the algorithm is therefore equivalent to the finite difference scheme

$$[D_t^+(u + \frac{1}{6}h^2D_xD_xu) = \alpha D_xD_xu + f]_i^n\,. \tag{227}$$

(More precisely, $Mc = b$ divided by h gives the equation above.)

Lumping the mass matrix. By applying Trapezoidal integration one can turn M into a diagonal matrix with $(h/2, h, \ldots, h, h/2)$ on the diagonal. Then there is no need to solve a linear system at each time level, and the finite element scheme becomes identical to a standard finite difference method

$$[D_t^+u = \alpha D_xD_xu + f]_i^n\,. \tag{228}$$

The Trapezoidal integration is not as accurate as exact integrat introduces therefore an error. Whether this error has a good or bad i on the overall numerical method is not immediately obvious, and is ana detail in Section 19.10. The effect of the error is at least not more seve what is produced by the finite difference method.

Making M diagonal is usually referred to as *lumping the mass matri* is an alternative method to using an integration rule based on the nod one can sum the entries in each row, place the sum on the diagonal, an other entries in the row equal to zero. For P1 elements the methods of the mass matrix give the same result.

19.7 Discretization in time by a Backward Euler sc

Time discretization. The Backward Euler scheme in time applie diffusion problem can be expressed as follows using the finite difference notation:

$$[D_t^-u = \alpha\nabla^2u + f(\boldsymbol{x}, t)]^n\,.$$

Written out, and collecting the unknown u^n on the left-hand side an known terms on the right-hand side, the time-discrete differential e becomes

$$u_{\mathrm{e}}^n - \Delta t\left(\alpha\nabla^2u_{\mathrm{e}}^n + f(\boldsymbol{x}, t_n)\right) = u_{\mathrm{e}}^{n-1}\,.$$

Equation (229) can compute $u_{\mathrm{e}}^1, u_{\mathrm{e}}^2, \ldots, u_{\mathrm{e}}^{N_t}$, if we have a start $u_{\mathrm{e}}^0 = I$ f initial condition. However, (229) is a partial differential equation in sp needs a solution method based on discretization in space. For this pur use an expansion as in (218)-(219).

Variational forms. Inserting (218)-(219) in (229), multiplying by $v \in V$), and integrating by parts, as we did in the Forward Euler case, re the variational form

$$\int_\Omega (u^nv + \Delta t\alpha\nabla u^n\cdot\nabla v)\,\mathrm{d}x = \int_\Omega u^{n-1}v\,\mathrm{d}x - \Delta t\int_\Omega f^nv\,\mathrm{d}x,\quad \forall v \in V$$

Expressed with u as u^n and u_1 as u^{n-1}, this becomes

$$\int_\Omega (uv + \Delta t\alpha\nabla u\cdot\nabla v)\,\mathrm{d}x = \int_\Omega u_1v\,\mathrm{d}x + \Delta t\int_\Omega f^nv\,\mathrm{d}x,$$

or with the more compact inner product notation,

$$(u, v) + \Delta t(\alpha\nabla u, \nabla v) = (u_1, v) + \Delta t(f^n, v)\,.$$

inear systems. Inserting $u = \sum_j c_j\psi_i$ and $u_1 = \sum_j c_{1,j}\psi_i$, and choosing v ɔ be the basis functions $\psi_i \in V$, $i = 0, \ldots, N$, together with doing some algebra, ɜad to the following linear system to be solved at each time level:

$$(M + \Delta t K)c = Mc_1 + f, \tag{233}$$

here M, K, and f are as in the Forward Euler case. This time we really have ɔ solve a linear system at each time level. The computational algorithm goes as ɔllows.

1. Compute M, K, and $A = M + \Delta t K$
2. Initialize u^0 by interpolation or projection
3. For $n = 1, 2, \ldots, N_t$:
   (a) compute $b = Mc_1 + f$
   (b) solve $Ac = b$
   (c) set $c_1 = c$

ɪ case of finite element basis functions, interpolation of the initial condition ɪ the nodes means $c_{1,j} = I(\boldsymbol{x}_j)$. Otherwise one has to solve the linear system $\sum_j \psi_j(x_i)c_j = I(x_i)$, where $\boldsymbol{x}_j$ denotes an interpolation point. Projection (or ɜalerkin's method) implies solving a linear system with M as coefficient matrix $\sum_j M_{i,j}c_{1,j} = (I, \psi_i)$, $i \in \mathcal{I}_s$.

We know what kind of finite difference operators the M and K matrices ɔrrespond to (after dividing by h), so (233) can be interpreted as the following nite difference method:

$$[D_t^-(u + \frac{1}{6}h^2 D_x D_x u) = \alpha D_x D_x u + f]_i^n\,. \tag{234}$$

The mass matrix M can be lumped, as explained in Section 19.6, and then ɪe linear system arising from the finite element method with P1 elements ɔrresponds to a plain Backward Euler finite difference method for the diffusion ɥuation:

$$[D_t^- u = \alpha D_x D_x u + f]_i^n\,. \tag{235}$$

9.8 Dirichlet boundary conditions

uppose now that the boundary condition (213) is replaced by a mixed Neumann ɪd Dirichlet condition,

$$u(\boldsymbol{x}, t) = u_0(\boldsymbol{x}, t), \qquad \boldsymbol{x} \in \partial\Omega_D, \tag{236}$$

$$-\alpha\frac{\partial}{\partial n}u(\boldsymbol{x}, t) = g(\boldsymbol{x}, t), \qquad \boldsymbol{x} \in \partial\Omega_N\,. \tag{237}$$

Using a Forward Euler discretization in time, the variational form a level becomes

$$\int_\Omega u^{n+1}v\,\mathrm{d}x = \int_\Omega (u^n - \Delta t\alpha\nabla u^n\cdot\nabla v)\,\mathrm{d}x - \Delta t\int_{\partial\Omega_N} gv\,\mathrm{d}s, \quad \forall v \in V$$

Boundary function. The Dirichlet condition $u = u_0$ at $\partial\Omega_D$ can be rated through a boundary function $B(\boldsymbol{x}) = u_0(\boldsymbol{x})$ and demanding that ɜ $\partial\Omega_D$. The expansion for u^n is written as

$$u^n(\boldsymbol{x}) = u_0(\boldsymbol{x}, t_n) + \sum_{j\in\mathcal{I}_s} c_j^n\psi_j(\boldsymbol{x})\,.$$

Inserting this expansion in the variational formulation and letting it hol basis functions ψ_i leads to the linear system

$$\sum_{j\in\mathcal{I}_s}\left(\int_\Omega \psi_i\psi_j\,\mathrm{d}x\right)c_j^{n+1} = \sum_{j\in\mathcal{I}_s}\left(\int_\Omega (\psi_i\psi_j - \Delta t\alpha\nabla\psi_i\cdot\nabla\psi_j)\,\mathrm{d}x\right)c_j^n -$$
$$\int_\Omega (u_0(\boldsymbol{x}, t_{n+1}) - u_0(\boldsymbol{x}, t_n) + \Delta t\alpha\nabla u_0(\boldsymbol{x}, t_n)\cdot$$
$$+ \Delta t\int_\Omega f\psi_i\,\mathrm{d}x - \Delta t\int_{\partial\Omega_N} g\psi_i\,\mathrm{d}s, \quad i \in \mathcal{I}_s\,.$$

In the following, we adopt the convention that the unknowns c_j^{n+1} are as c_j, while the known c_j^n from the previous time level are denoted by ɔ

Finite element basis functions. When using finite elements, eac function φ_i is associated with a node x_i. We have a collection of nodes on the boundary $\partial\Omega_D$. Suppose U_k^n is the known Dirichlet value at x_k t_n ($U_k^n = u_0(x_k, t_n)$). The appropriate boundary function is then

$$B(\boldsymbol{x}, t_n) = \sum_{j\in I_b} U_j^n\varphi_j\,.$$

The unknown coefficients c_j are associated with the rest of the nodes have numbers $\nu(i)$, $i \in \mathcal{I}_s = \{0, \ldots, N\}$. The basis functions for V are cl $\psi_i = \varphi_{\nu(i)}$, $i \in \mathcal{I}_s$, and all of these vanish at the boundary nodes as they The expansion for u^{n+1} and u^n become

$$u^n = \sum_{j\in I_b} U_j^n\varphi_j + \sum_{j\in\mathcal{I}_s} c_{1,j}\varphi_{\nu(j)},$$
$$u^{n+1} = \sum_{j\in I_b} U_j^{n+1}\varphi_j + \sum_{j\in\mathcal{I}_s} c_j\varphi_{\nu(j)}\,.$$

he equations for the unknown coefficients c_i become

$$\sum_{j\in\mathcal{I}_s}\left(\int_\Omega \varphi_i\varphi_j\,\mathrm{d}x\right)c_j = \sum_{j\in\mathcal{I}_s}\left(\int_\Omega (\varphi_i\varphi_j - \Delta t\alpha\nabla\varphi_i\cdot\nabla\varphi_j)\,\mathrm{d}x\right)c_{1,j} - \sum_{j\in I_b}\int_\Omega \left(\varphi_i\varphi_j(U_j^{n+1}-U_j^n) + \Delta t\alpha\nabla\varphi_i\cdot\nabla\varphi_j U_j^n\right)\mathrm{d}x + \Delta t\int_\Omega f\varphi_i\,\mathrm{d}x - \Delta t\int_{\partial\Omega_N} g\varphi_i\,\mathrm{d}s,\quad i\in\mathcal{I}_s\,.$$

Modification of the linear system. Instead of introducing a boundary unction B we can work with basis functions associated with all the nodes and ncorporate the Dirichlet conditions by modifying the linear system. Let $\mathcal{I}_s$ be ne index set that counts all the nodes: $\{0,1,\ldots,N=N_n\}$. The expansion for ^{n} is then $\sum_{j\in\mathcal{I}_s} c_j^n\varphi_j$ and the variational form becomes

$$\sum_{j\in\mathcal{I}_s}\left(\int_\Omega \varphi_i\varphi_j\,\mathrm{d}x\right)c_j = \sum_{j\in\mathcal{I}_s}\left(\int_\Omega (\varphi_i\varphi_j - \Delta t\alpha\nabla\varphi_i\cdot\nabla\varphi_j)\,\mathrm{d}x\right)c_{1,j} - \Delta t\int_\Omega f\varphi_i\,\mathrm{d}x - \Delta t\int_{\partial\Omega_N} g\varphi_i\,\mathrm{d}s\,.$$

We introduce the matrices M and K with entries $M_{i,j}=\int_\Omega\varphi_i\varphi_j\,\mathrm{d}x$ and $K_{i,j}=\int_\Omega\alpha\nabla\varphi_i\cdot\nabla\varphi_j\,\mathrm{d}x$, respectively. In addition, we define the vectors c, c_1, and f with entries c_i, $c_{1,i}$, and $\int_\Omega f\varphi_i\,\mathrm{d}x - \int_{\partial\Omega_N} g\varphi_i\,\mathrm{d}s$. The equation system can then be written as

$$Mc = Mc_1 - \Delta tKc_1 + \Delta tf\,. \tag{239}$$

When M, K, and b are assembled without paying attention to Dirichlet boundary conditions, we need to replace equation k by $c_k = U_k$ for k corresponding to all boundary nodes ($k\in I_b$). The modification of M consists in setting $M_{k,j}=0$, $j\in\mathcal{I}_s$, and the $M_{k,k}=1$. Alternatively, a modification that preserves the symmetry of M can be applied. At each time level one forms $b = Mc_1 - \Delta tKc_1 + \Delta tf$ and sets $b_k = U_k^{n+1}$, $k\in I_b$, and solves the system $Ic = b$.

In case of a Backward Euler method, the system becomes (233). We can write the system as $Ac = b$, with $A = M + \Delta tK$ and $b = Mc_1 + f$. Both M and K needs to be modified because of Dirichlet boundary conditions, but the diagonal entries in K should be set to zero and those in M to unity. In this way, $A_{k,k}=1$. The right-hand side must read $b_k = U_k^n$ for $k\in I_b$ (assuming the unknown is sought at time level t_n).

9.9 Example: Oscillating Dirichlet boundary condition

We shall address the one-dimensional initial-boundary value problem

$$\begin{aligned} u_t &= (\alpha u_x)_x + f, && \boldsymbol{x}\in\Omega=[0,L],\ t\in(0,T],\\ u(x,0) &= 0, && \boldsymbol{x}\in\Omega,\\ u(0,t) &= a\sin\omega t, && t\in(0,T],\\ u_x(L,t) &= 0, && t\in(0,T]\,. \end{aligned}$$

A physical interpretation may be that u is the temperature deviatio a constant mean temperature in a body Ω that is subject to an osc temperature (e.g., day and night, or seasonal, variations) at $x=0$.

We use a Backward Euler scheme in time and P1 elements of constan h in space. Incorporation of the Dirichlet condition at $x=0$ through m the linear system at each time level means that we carry out the compu as explained in Section 19.7 and get a system (233). The M and K r computed without paying attention to Dirichlet boundary conditions b

$$M = \frac{h}{6}\begin{pmatrix} 2 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0\\ 1 & 4 & 1 & \ddots & & & & & \vdots\\ 0 & 1 & 4 & 1 & \ddots & & & & \vdots\\ \vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots\\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots\\ \vdots & & & 0 & 1 & 4 & 1 & \ddots & \vdots\\ \vdots & & & & \ddots & \ddots & \ddots & \ddots & 0\\ \vdots & & & & & \ddots & 1 & 4 & 1\\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 1 & 2 \end{pmatrix}$$

$$K = \frac{\alpha}{h}\begin{pmatrix} 1 & -1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0\\ -1 & 2 & -1 & \ddots & & & & & \vdots\\ 0 & -1 & 2 & -1 & \ddots & & & & \vdots\\ \vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots\\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots\\ \vdots & & & 0 & -1 & 2 & -1 & \ddots & \vdots\\ \vdots & & & & \ddots & \ddots & \ddots & \ddots & 0\\ \vdots & & & & & \ddots & -1 & 2 & -1\\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & -1 & 1 \end{pmatrix}$$

The right-hand side of the variational form contains Mc_1 since there is n term (f) and no boundary term from the integration by parts ($u_x=0$ a

nd we compute as if $u_x = 0$ at $x = 0$ too). We must incorporate the Dirichlet oundary condition $c_0 = a\sin\omega t_n$ by ensuring that this is the first equation in ıe linear system. To this end, the first row in K and M are set to zero, but ıe diagonal entry $M_{0,0}$ is set to 1. The right-hand side is $b = Mc_1$, and we set $) = a\sin\omega t_n$. Note that in this approach, $N = N_n$, and c equals the unknown at each node in the mesh. We can write the complete linear system as

$$c_0 = a\sin\omega t_n, \tag{246}$$

$$(c_{i-1} + 4c_i + c_{i+1}) + \Delta t\frac{\alpha}{h}(-c_{i-1} + 2c_i + c_{i+1}) = \frac{h}{6}(c_{1,i-1} + 4c_{1,i} + c_{1,i+1}), \tag{247}$$

$$i = 1, \ldots, N_n - 1,$$

$$\frac{h}{6}(c_{i-1} + 2c_i) + \Delta t\frac{\alpha}{h}(-c_{i-1} + c_i) = \frac{h}{6}(c_{1,i-1} + 2c_{1,i}), \quad i = N_n\,. \tag{248}$$

The Dirichlet boundary condition can alternatively be implemented through boundary function $B(x,t) = a\sin\omega t\varphi_0(x)$:

$$u^n(x) = a\sin\omega t_n\varphi_0(x) + \sum_{j\in\mathcal{I}_s} c_j\varphi_{\nu(j)}(x), \quad \nu(j) = j+1\,.$$

ow, $N = N_n - 1$ and the c vector contains values of u at nodes $1, 2, \ldots, N_n$. he right-hand side gets a contribution

$$\int_0^L (a(\sin\omega t_n - \sin\omega t_{n-1})\varphi_0\varphi_i - \Delta t\alpha a\sin\omega t_n\nabla\varphi_0\cdot\nabla\varphi_i)\,\mathrm{d}x\,. \tag{249}$$

9.10 Analysis of the discrete equations

he diffusion equation $u_t = \alpha u_{xx}$ allows a (Fourier) wave component $u = \exp(\beta t + ikx)$ as solution if $\beta = -\alpha k^2$, which follows from inserting the wave omponent in the equation. The exact wave component can alternatively be ritten as

$$u = A_\mathrm{e}^n e^{ikx}, \quad A_\mathrm{e} = e^{-\alpha k^2\Delta t}\,. \tag{250}$$

Many numerical schemes for the diffusion equation has a similar wave component s solution:

$$u_q^n = A^n e^{ikx}, \tag{251}$$

here is an amplification factor to be calculated by inserting (252) in the scheme. We introduce $x = qh$, or $x = q\Delta x$ to align the notation with that frequently sed in finite difference methods.

A convenient start of the calculations is to establish some results for finite difference operators acting on

$$u_q^n = A^n e^{ikq\Delta x}\,.$$

$$[D_t^+ A^n e^{ikq\Delta x}]^n = A^n e^{ikq\Delta x}\frac{A-1}{\Delta t},$$
$$[D_t^- A^n e^{ikq\Delta x}]^n = A^n e^{ikq\Delta x}\frac{1-A^{-1}}{\Delta t},$$
$$[D_t A^n e^{ikq\Delta x}]^{n+\frac{1}{2}} = A^{n+\frac{1}{2}}e^{ikq\Delta x}\frac{A^{\frac{1}{2}} - A^{-\frac{1}{2}}}{\Delta t} = A^n e^{ikq\Delta x}\frac{A-1}{\Delta t},$$
$$[D_x D_x A^n e^{ikq\Delta x}]_q = -A^n\frac{4}{\Delta x^2}\sin^2\left(\frac{k\Delta x}{2}\right)\,.$$

Forward Euler discretization. We insert (252) in the Forward Euler with P1 elements in space and $f = 0$ (this type of analysis can only be out if $f = 0$),

$$[D_t^+(u + \frac{1}{6}h^2 D_x D_x u) = \alpha D_x D_x u]_q^n\,.$$

We have

$$[D_t^+ D_x D_x A e^{ikx}]_q^n = [D_t^+ A]^n [D_x D_x e^{ikx}]_q = -A^n e^{ikp\Delta x}\frac{A-1}{\Delta t}\frac{4}{\Delta x^2}\sin^2($$

The term $[D_t^+ Ae^{ikx} + \frac{1}{6}\Delta x^2 D_t^+ D_x D_x Ae^{ikx}]_q^n$ then reduces to

$$\frac{A-1}{\Delta t} - \frac{1}{6}\Delta x^2\frac{A-1}{\Delta t}\frac{4}{\Delta x^2}\sin^2(\frac{k\Delta x}{2}),$$

or

$$\frac{A-1}{\Delta t}\left(1 - \frac{2}{3}\sin^2(k\Delta x/2)\right)\,.$$

Introducing $p = k\Delta x/2$ and $C = \alpha\Delta t/\Delta x^2$, the complete scheme becor

$$(A-1)\left(1 - \frac{2}{3}\sin^2 p\right) = -4C\sin^2 p,$$

from which we find A to be

$$A = 1 - 4C\frac{\sin^2 p}{1 - \frac{2}{3}\sin^2 p}\,.$$

How does this A change the stability criterion compared to the I Euler finite difference scheme and centered differences in space? The s criterion is $|A| \leq 1$, which here implies $A \leq 1$ and $A \geq -1$. The former i fulfilled, while the latter leads to

$$4C\frac{\sin^2 p}{1+\frac{2}{3}\sin^2 p} \leq 2\,.$$

he factor $\sin^2 p/(1-\frac{2}{3}\sin^2 p)$ can be plotted for $p \in [0,\pi/2]$, and the maximum alue goes to 3 as $p \to \pi/2$. The worst case for stability therefore occurs for the ıortest possible wave, $p = \pi/2$, and the stability criterion becomes

$$C \leq \frac{1}{6} \quad \Rightarrow \quad \Delta t \leq \frac{\Delta x^2}{6\alpha}, \tag{254}$$

hich is a factor 1/3 worse than for the standard Forward Euler finite difference ıethod for the diffusion equation, which demands $C \leq 1/2$. Lumping the mass ıatrix will, however, recover the finite difference method and therefore imply $' \leq 1/2$ for stability.

ackward Euler discretization. We can use the same approach and insert 52) in the Backward Euler scheme with P1 elements in space and $f = 0$:

$$[D_t^-(u + \frac{1}{6}h^2 D_x D_x u) = \alpha D_x D_x u]_i^n\,. \tag{255}$$

imilar calculations as in the Forward Euler case lead to

$$(1 - A^{-1})\left(1 - \frac{2}{3}\sin^2 p\right) = -4C\sin^2 p,$$

nd hence

$$A = \left(1 + 4C\frac{\sin^2 p}{1-\frac{2}{3}\sin^2 p}\right)^{-1}\,.$$

omparing amplification factors. It is of interest to compare A and A_e as ınctions of p for some C values. Figure 48 display the amplification factors ›r the Backward Euler scheme corresponding a coarse mesh with $C = 2$ and a ıesh at the stability limit of the Forward Euler scheme in the finite difference ıethod, $C = 1/2$. Figures 49 and 50 shows how the accuracy increases with ›wer C values for both the Forward Euler and Backward schemes, respectively. he striking fact, however, is that the accuracy of the finite element method significantly less than the finite difference method for the same value of C. umping the mass matrix to recover the numerical amplification factor A of the nite difference method is therefore a good idea in this problem.

Remaining tasks:

- Taylor expansion of the error in the amplification factor $A_\mathrm{e} - A$
- Taylor expansion of the error $e = (A_\mathrm{e}^n - A^n)e^{ikx}$
- L^2 norm of e

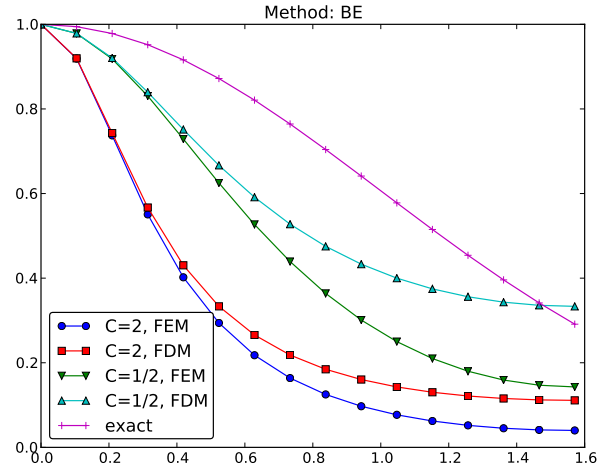


Figure 48: Comparison of coarse-mesh amplification factors for Backwa discretization of a 1D diffusion equation.

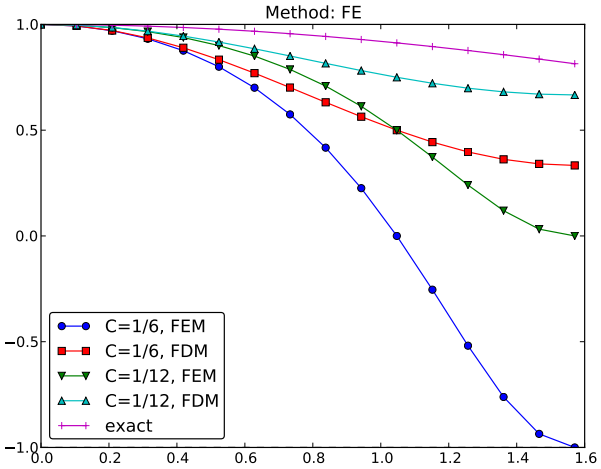
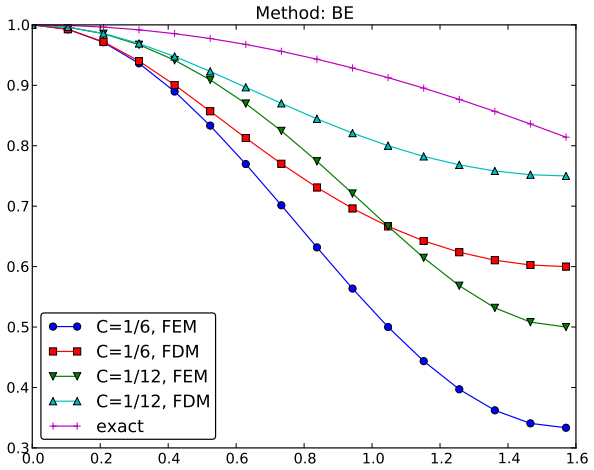


Figure 49: Comparison of fine-mesh amplification factors for Forwa discretization of a 1D diffusion equation.



igure 50: Comparison of fine-mesh amplification factors for Backward Euler iscretization of a 1D diffusion equation.

20 Systems of differential equations

Many mathematical models involve $m+1$ unknown functions govern system of $m+1$ differential equations. In abstract form we may der unknowns by $u^{(0)},\ldots,u^{(m)}$ and write the governing equations as

$$\mathcal{L}_0(u^{(0)},\ldots,u^{(m)}) = 0,$$
$$\vdots$$
$$\mathcal{L}_m(u^{(0)},\ldots,u^{(m)}) = 0,$$

where $\mathcal{L}_i$ is some differential operator defining differential equation nun

20.1 Variational forms

There are basically two ways of formulating a variational form for a sy differential equations. The first method treats each equation independ a scalar equation, while the other method views the total system as a equation with a vector function as unknown.

Let us start with the one equation at a time approach. We multiply e number i by some test function $v^{(i)} \in V^{(i)}$ and integrate over the doma

$$\int_\Omega \mathcal{L}^{(0)}(u^{(0)},\ldots,u^{(m)})v^{(0)}\,\mathrm{d}x = 0,$$
$$\vdots$$
$$\int_\Omega \mathcal{L}^{(m)}(u^{(0)},\ldots,u^{(m)})v^{(m)}\,\mathrm{d}x = 0\,.$$

Terms with second-order derivatives may be integrated by parts, with N conditions inserted in boundary integrals. Let

$$V^{(i)} = \mathrm{span}\{\psi_0^{(i)},\ldots,\psi_{N_i}^{(i)}\},$$

such that

$$u^{(i)} = B^{(i)}(\boldsymbol{x}) + \sum_{j=0}^{N_i} c_j^{(i)}\psi_j^{(i)}(\boldsymbol{x}),$$

where $B^{(i)}$ is a boundary function to handle nonzero Dirichlet conditions. that different unknowns live in different spaces with different basis functi numbers of degrees of freedom.

From the m equations in the variational forms we can derive m systems of algebraic equations for the $\Pi_{i=0}^m N_i$ unknown coefficients c $0,\ldots,N_i$, $i=0,\ldots,m$.

The alternative method for deriving a variational form for a system of ifferential equations introduces a vector of unknown functions

$$\boldsymbol{u} = (u^{(0)}, \ldots, u^{(m)}),$$

vector of test functions

$$\boldsymbol{v} = (u^{(0)}, \ldots, u^{(m)}),$$

ith

$$\boldsymbol{u}, \boldsymbol{v} \in \boldsymbol{V} = V^{(0)} \times \cdots \times V^{(m)}\,.$$

/ith nonzero Dirichlet conditions, we have a vector $\boldsymbol{B} = (B^{(0)}, \ldots, B^{(m)})$ with oundary functions and then it is $\boldsymbol{u} - \boldsymbol{B}$ that lies in $\boldsymbol{V}$, not $\boldsymbol{u}$ itself.

The governing system of differential equations is written

$$\boldsymbol{\mathcal{L}}(\boldsymbol{u}) = 0,$$

here

$$\boldsymbol{\mathcal{L}}(\boldsymbol{u}) = (\mathcal{L}^{(0)}(\boldsymbol{u}), \ldots, \mathcal{L}^{(m)}(\boldsymbol{u}))\,.$$

he variational form is derived by taking the inner product of the vector of quations and the test function vector:

$$\int_\Omega \boldsymbol{\mathcal{L}}(\boldsymbol{u}) \cdot \boldsymbol{v} = 0 \quad \forall \boldsymbol{v} \in \boldsymbol{V}\,. \tag{259}$$

Observe that (259) is one scalar equation. To derive systems of algebraic quations for the unknown coefficients in the expansions of the unknown functions, one chooses m linearly independent $\boldsymbol{v}$ vectors to generate m independent ariational forms from (259). The particular choice $\boldsymbol{v} = (v^{(0)}, 0, \ldots, 0)$ recovers 256), $\boldsymbol{v} = (0, \ldots, 0, v^{(m)}$ recovers (258), and $\boldsymbol{v} = (0, \ldots, 0, v^{(i)}, 0, \ldots, 0)$ recovers ne variational form number i, $\int_\Omega \mathcal{L}^{(i)} v^{(i)}\,\mathrm{d}x = 0$, in (256)-(258).

0.2 A worked example

/e now consider a specific system of two partial differential equations in two pace dimensions:

$$\mu\nabla^2 w = -\beta, \tag{260}$$
$$\kappa\nabla^2 T = -\mu||\nabla w||^2\,. \tag{261}$$

he unknown functions $w(x,y)$ and $T(x,y)$ are defined in a domain Ω, while , β, and κ are given constants. The norm in (261) is the standard Eucledian orm:

$$||\nabla w||^2 = \nabla w \cdot \nabla w = w_x^2 + w_y^2\,.$$

The boundary conditions associated with (260)-(261) are $w = 0$ on $T = T_0$ on $\partial\Omega$. Each of the equations (260) and (261) need one cond each point on the boundary.

The system (260)-(261) arises from fluid flow in a straight pipe, with tl in the direction of the pipe. The domain Ω is a cross section of the pipe, velocity in the z direction, μ is the viscosity of the fluid, β is the pressure along the pipe, T is the temperature, and κ is the heat conduction co of the fluid. The equation (260) comes from the Navier-Stokes equatic (261) follows from the energy equation. The term $-\mu||\nabla w||^2$ models he the fluid due to internal friction.

Observe that the system (260)-(261) has only a one-way coupling: T on w, but w does not depend on T, because we can solve (260) with to w and then (261) with respect to T. Some may argue that this real system of PDEs, but just two scalar PDEs. Nevertheless, the coupling is convenient when comparing different variational forms and implementations.

20.3 Identical function spaces for the unknowns

Let us first apply the same function space V for w and T (or more p $w \in V$ and $T - T_0 \in V$). With

$$V = \mathrm{span}\{\psi_0(x,y), \ldots, \psi_N(x,y)\},$$

we write

$$w = \sum_{j=0}^{N} c_j^{(w)}\psi_j, \quad T = T_0 + \sum_{j=0}^{N} c_j^{(T)}\psi_j\,.$$

Note that w and T in (260)-(261) denote the exact solution of the PDE w and T (262) are the discrete functions that approximate the exact s It should be clear from the context whether a symbol means the e approximate solution, but when we need both at the same time, w subscript e to denote the exact solution.

Variational form of each individual PDE. Inserting the expansio in the governing PDEs, results in a residual in each equation,

$$R_w = \mu\nabla^2 w + \beta,$$
$$R_T = \kappa\nabla^2 T + \mu||\nabla w||^2\,.$$

A Galerkin method demands R_w and R_T do be orthogonal to V:

$$\int_\Omega R_w v \,\mathrm{d}x = 0 \quad \forall v \in V,$$
$$\int_\Omega R_T v \,\mathrm{d}x = 0 \quad \forall v \in V\,.$$

ecause of the Dirichlet conditions, $v = 0$ on $\partial\Omega$. We integrate the Laplace erms by parts and note that the boundary terms vanish since $v = 0$ on $\partial\Omega$:

$$\int_\Omega \mu\nabla w\cdot\nabla v \,\mathrm{d}x = \int_\Omega \beta v \,\mathrm{d}x \quad \forall v \in V, \tag{265}$$
$$\int_\Omega \kappa\nabla T\cdot\nabla v \,\mathrm{d}x = \int_\Omega \mu\nabla w\cdot\nabla w\, v \,\mathrm{d}x \quad \forall v \in V\,. \tag{266}$$

ompound scalar variational form. The alternative way of deriving the ariational from is to introduce a test vector function $\boldsymbol{v} \in \boldsymbol{V} = V \times V$ and take ie inner product of $\boldsymbol{v}$ and the residuals, integrated over the domain:

$$\int_\Omega (R_w, R_T)\cdot\boldsymbol{v} \,\mathrm{d}x = 0 \quad \forall \boldsymbol{v} \in \boldsymbol{V}\,.$$

/ith $\boldsymbol{v} = (v_0, v_1)$ we get

$$\int_\Omega (R_w v_0 + R_T v_1) \,\mathrm{d}x = 0 \quad \forall \boldsymbol{v} \in \boldsymbol{V}\,.$$

ıtegrating the Laplace terms by parts results in

$$\int_\Omega (\mu\nabla w\cdot\nabla v_0 + \kappa\nabla T\cdot\nabla v_1) \,\mathrm{d}x = \int_\Omega (\beta v_0 + \mu\nabla w\cdot\nabla w\, v_1) \,\mathrm{d}x, \quad \forall \boldsymbol{v} \in \boldsymbol{V}\,. \tag{267}$$

hoosing $v_0 = v$ and $v_1 = 0$ gives the variational form (265), while $v_0 = 0$ and $_1 = v$ gives (266).

With the inner product notation, $(p, q) = \int_\Omega pq \,\mathrm{d}x$, we can alternatively write 65) and (266) as

$$(\mu\nabla w, \nabla v) = (\beta, v) \quad \forall v \in V,$$
$$(\kappa\nabla T, \nabla v) = (\mu\nabla w\cdot\nabla w, v) \quad \forall v \in V,$$

r since μ and κ are considered constant,

$$\mu(\nabla w, \nabla v) = (\beta, v) \quad \forall v \in V, \tag{268}$$
$$\kappa(\nabla T, \nabla v) = \mu(\nabla w\cdot\nabla w, v) \quad \forall v \in V\,. \tag{269}$$

Decoupled linear systems. The linear systems governing the coe $c_j^{(w)}$ and $c_j^{(T)}$, $j = 0, \ldots, N$, are derived by inserting the expansions ((265) and (266), and choosing $v = \psi_i$ for $i = 0, \ldots, N$. The result beco

$$\sum_{j=0}^{N} A_{i,j}^{(w)} c_j^{(w)} = b_i^{(w)}, \quad i = 0, \ldots, N,$$
$$\sum_{j=0}^{N} A_{i,j}^{(T)} c_j^{(T)} = b_i^{(T)}, \quad i = 0, \ldots, N,$$
$$A_{i,j}^{(w)} = \mu(\nabla\psi_j, \nabla\psi_i),$$
$$b_i^{(w)} = (\beta, \psi_i),$$
$$A_{i,j}^{(T)} = \kappa(\nabla\psi_j, \nabla\psi_i),$$
$$b_i^{(T)} = \mu((\sum_j c_j^{(w)}\nabla\psi_j)\cdot(\sum_k c_k^{(w)}\nabla\psi_k), \psi_i)\,.$$

It can also be instructive to write the linear systems using matri vectors. Define K as the matrix corresponding to the Laplace operator ∇ is, $K_{i,j} = (\nabla\psi_j, \nabla\psi_i)$. Let us introduce the vectors

$$b^{(w)} = (b_0^{(w)}, \ldots, b_N^{(w)}),$$
$$b^{(T)} = (b_0^{(T)}, \ldots, b_N^{(T)}),$$
$$c^{(w)} = (c_0^{(w)}, \ldots, c_N^{(w)}),$$
$$c^{(T)} = (c_0^{(T)}, \ldots, c_N^{(T)})\,.$$

The system (270)-(271) can now be expressed in matrix-vector form as

$$\mu K c^{(w)} = b^{(w)},$$
$$\kappa K c^{(T)} = b^{(T)}\,.$$

We can solve the first system for $c^{(w)}$, and then the right-hand sid known such that we can solve the second system for $c^{(T)}$.

Coupled linear systems. Despite the fact that w can be comput without knowing T, we shall now pretend that w and T enter a two-way c such that we need to derive the algebraic equations as *one system* fo unknowns $c_j^{(w)}$ and $c_j^{(T)}$, $j = 0, \ldots, N$. This system is nonlinear in $c_j^{(w)}$ of the $\nabla w\cdot\nabla w$ product. To remove this nonlinearity, imagine that we i an iteration method where we replace $\nabla w\cdot\nabla w$ by $\nabla w_-\cdot\nabla w$, w_- bein computed in the previous iteration. Then the term $\nabla w_-\cdot\nabla w$ is line since w_- is known. The total linear system becomes

$$\sum_{j=0}^{N} A_{i,j}^{(w,w)} c_j^{(w)} + \sum_{j=0}^{N} A_{i,j}^{(w,T)} c_j^{(T)} = b_i^{(w)}, \quad i = 0, \ldots, N, \tag{278}$$

$$\sum_{j=0}^{N} A_{i,j}^{(T,w)} c_j^{(w)} + \sum_{j=0}^{N} A_{i,j}^{(T,T)} c_j^{(T)} = b_i^{(T)}, \quad i = 0, \ldots, N, \tag{279}$$

$$A_{i,j}^{(w,w)} = \mu(\nabla\psi_j, \psi_i), \tag{280}$$

$$A_{i,j}^{(w,T)} = 0, \tag{281}$$

$$b_i^{(w)} = (\beta, \psi_i), \tag{282}$$

$$A_{i,j}^{(w,T)} = \mu((\nabla\psi w_-) \cdot \nabla\psi_j), \psi_i), \tag{283}$$

$$A_{i,j}^{(T,T)} = \kappa(\nabla\psi_j, \psi_i), \tag{284}$$

$$b_i^{(T)} = 0\,. \tag{285}$$

his system can alternatively be written in matrix-vector form as

$$\mu K c^{(w)} = 0 b^{(w)}, \tag{286}$$

$$L c^{(w)} + \kappa K c^{(T)} = 0, \tag{287}$$

ith L as the matrix from the $\nabla w_- \cdot \nabla$ operator: $L_{i,j} = A_{i,j}^{(w,T)}$.

The matrix-vector equations are often conveniently written in block form:

$$\begin{pmatrix} \mu K & 0 \\ L & \kappa K \end{pmatrix} \begin{pmatrix} c^{(w)} \\ c^{(T)} \end{pmatrix} = \begin{pmatrix} b^{(w)} \\ 0 \end{pmatrix},$$

Note that in the general case where all unknowns enter all equations, we ave to solve the compound system (297)-(298) since then we cannot utilize the ɔecial property that (270) does not involve T and can be solved first.

When the viscosity depends on the temperature, the $\mu\nabla^2 w$ term must be eplaced by $\nabla \cdot (\mu(T)\nabla w)$, and then T enters the equation for w. Now we have two-way coupling since both equations contain w and T and therefore must e solved simultaneously Th equation $\nabla \cdot (\mu(T)\nabla w) = -\beta$ is nonlinear, and if ɔme iteration procedure is invoked, where we use a previously computed T_- in ıe viscosity ($\mu(T_-)$), the coefficient is known, and the equation involves only ne unknown, w. In that case we are back to the one-way coupled set of PDEs.

We may also formulate our PDE system as a vector equation. To this end, we ıtroduce the vector of unknowns $\boldsymbol{u} = (u^{(0)}, u^{(1)})$, where $u^{(0)} = w$ and $u^{(1)} = T$. Ve then have

$$\nabla^2 \boldsymbol{u} = \begin{pmatrix} -\mu^{-1}\beta \\ -\kappa^{-1}\mu\nabla u^{(0)} \cdot \nabla u^{(0)} \end{pmatrix}.$$

20.4 Different function spaces for the unknowns

It is easy to generalize the previous formulation to the case where $w \in V$ $T \in V^{(T)}$, where $V^{(w)}$ and $V^{(T)}$ can be different spaces with different n of degrees of freedom. For example, we may use quadratic basis funct w and linear for T. Approximation of the unknowns by different finite spaces is known as *mixed finite element methods*.

We write

$$V^{(w)} = \text{span}\{\psi_0^{(w)}, \ldots, \psi_{N_w}^{(w)}\},$$
$$V^{(T)} = \text{span}\{\psi_0^{(T)}, \ldots, \psi_{N_T}^{(T)}\}\,.$$

The next step is to multiply (260) by a test function $v^{(w)} \in V^{(w)}$ and (2 $v^{(T)} \in V^{(T)}$, integrate by parts and arrive at

$$\int_\Omega \mu\nabla w \cdot \nabla v^{(w)}\,\mathrm{d}x = \int_\Omega \beta v^{(w)}\,\mathrm{d}x \quad \forall v^{(w)} \in V^{(w)},$$
$$\int_\Omega \kappa\nabla T \cdot \nabla v^{(T)}\,\mathrm{d}x = \int_\Omega \mu\nabla w \cdot \nabla w\, v^{(T)}\,\mathrm{d}x \quad \forall v^{(T)} \in V^{(T)}\,.$$

The compound scalar variational formulation applies a test vector f $\boldsymbol{v} = (v^{(w)}, v^{(T)})$ and reads

$$\int_\Omega (\mu\nabla w \cdot \nabla v^{(w)} + \kappa\nabla T \cdot \nabla v^{(T)})\,\mathrm{d}x = \int_\Omega (\beta v^{(w)} + \mu\nabla w \cdot \nabla w\, v^{(T)})\,\mathrm{d}x$$

valid $\forall \boldsymbol{v} \in \boldsymbol{V} = V^{(w)} \times V^{(T)}$.

The associated linear system is similar to (270)-(271) or (297)-(298) that we need to distinguish between $\psi_i^{(w)}$ and $\psi_i^{(T)}$, and the range in tl over j must match the number of degrees of freedom in the spaces V $V^{(T)}$. The formulas become

$$\sum_{j=0}^{N_w} A_{i,j}^{(w)} c_j^{(w)} = b_i^{(w)}, \quad i = 0, \ldots, N_w,$$

$$\sum_{j=0}^{N_T} A_{i,j}^{(T)} c_j^{(T)} = b_i^{(T)}, \quad i = 0, \ldots, N_T,$$

$$A_{i,j}^{(w)} = \mu(\nabla\psi_j^{(w)}, \psi_i^{(w)}),$$
$$b_i^{(w)} = (\beta, \psi_i^{(w)}),$$
$$A_{i,j}^{(T)} = \kappa(\nabla\psi_j^{(T)}, \psi_i^{(T)}),$$
$$b_i^{(T)} = \mu(\nabla w_-, \psi_i^{(T)})\,.$$

In the case we formulate one compound linear system involving both $c_j^{(w)}$, $= 0, \ldots, N_w$, and $c_j^{(T)}$, $j = 0, \ldots, N_T$, (297)-(298) becomes

$$\sum_{j=0}^{N_w} A_{i,j}^{(w,w)} c_j^{(w)} + \sum_{j=0}^{N_T} A_{i,j}^{(w,T)} c_j^{(T)} = b_i^{(w)}, \quad i = 0, \ldots, N_w, \tag{297}$$

$$\sum_{j=0}^{N_w} A_{i,j}^{(T,w)} c_j^{(w)} + \sum_{j=0}^{N_T} A_{i,j}^{(T,T)} c_j^{(T)} = b_i^{(T)}, \quad i = 0, \ldots, N_T, \tag{298}$$

$$A_{i,j}^{(w,w)} = \mu(\nabla\psi_j^{(w)}, \psi_i^{(w)}), \tag{299}$$

$$A_{i,j}^{(w,T)} = 0, \tag{300}$$

$$b_i^{(w)} = (\beta, \psi_i^{(w)}), \tag{301}$$

$$A_{i,j}^{(w,T)} = \mu(\nabla w_- \cdot \nabla\psi_j^{(w)}), \psi_i^{(T)}), \tag{302}$$

$$A_{i,j}^{(T,T)} = \kappa(\nabla\psi_j^{(T)}, \psi_i^{(T)}), \tag{303}$$

$$b_i^{(T)} = 0\,. \tag{304}$$

he corresponding block form

$$\begin{pmatrix} \mu K^{(w)} & 0 \\ L & \kappa K^{(T)} \end{pmatrix} \begin{pmatrix} c^{(w)} \\ c^{(T)} \end{pmatrix} = \begin{pmatrix} b^{(w)} \\ 0 \end{pmatrix},$$

as square and rectangular block matrices: $K^{(w)}$ is $N_w \times N_w$, $K^{(T)}$ is $N_T \times N_T$, hile L is $N_T \times N_w$,

0.5 Computations in 1D

/e can reduce the system (260)-(261) to one space dimension, which corresponds o flow in a channel between two flat plates. Alternatively, one may consider ow in a circular pipe, introduce cylindrical coordinates, and utilize the radial ymmetry to reduce the equations to a one-dimensional problem in the radial oordinate. The former model becomes

$$\mu w_{xx} = -\beta, \tag{305}$$

$$\kappa T_{xx} = -\mu w_x^2, \tag{306}$$

hile the model in the radial coordinate r reads

$$\mu \frac{1}{r}\frac{d}{dr}\left(r\frac{dw}{dr}\right) = -\beta, \tag{307}$$

$$\kappa \frac{1}{r}\frac{d}{dr}\left(r\frac{dT}{dr}\right) = -\mu\left(\frac{dw}{dr}\right)^2. \tag{308}$$

The domain for (305)-(306) is $\Omega = [0, H]$, with boundary conditions $w(H) = 0$ and $T(0) = T(H) = T_0$. For (307)-(308) the domain is $[0, R]$ (the radius of the pipe) and the boundary conditions are $du/dr = dT/dr$ $r = 0$, $u(R) = 0$, and $T(R) = T_0$.

Calculations to be continued...

21 Exercises

Exercise 23: Refactor functions into a more general

Section 11.2 displays three functions for computing the analytical sol some simple model problems. There is quite some repetitive code, sug that the functions can benefit from being refactored into a class where can define the $f(x)$, $a(x)$, and the boundary conditions in particular r in subclasses. Demonstrate how the new class can be used to solve tl particular problems in Section 11.2.

In the method that computes the solution, check that the solution foun the differential equation and the boundary conditions. Filename: `uxx_f_s`

Exercise 24: Compute the deflection of a cable wit functions

A hanging cable of length L with significant tension has a downward de $w(x)$ governed by

Solve

$$Tw''(x) = \ell(x),$$

where T is the tension in the cable and $\ell(x)$ the load per unit length. T is fixed at $x = 0$ and $x = L$ so the boundary conditions become $T(0) = T$ We assume a constant load $\ell(x) = \text{const}$.

The solution is expected to be symmetric around $x = L/2$. Formula problem for $x \in \Omega = [0, L/2]$ and then scaling it, results in the scaled p for the dimensionless vertical deflection u:

$$u'' = 1, \quad x \in (0,1), \quad u(0) = 0,\ u'(1) = 0\,.$$

Introduce the function space spanned by $\psi_i = \sin((i+1)\pi x/2)$, $i = 1$ Use a Galerkin and a least squares method to find the coefficients c_j in $\sum_j c_j\psi_j$. Find how fast the coefficients decrease in magnitude by loo c_j/c_{j-1}. Find the error in the maximum deflection at $x = 1$ when only o function is used ($N = 0$).

What happens if we choose basis functions $\psi_i = \sin((i+1)\pi x)$? The functions are appropriate if we do not utilize symmetry and solve the p on $[0, L]$. A scaled version of this problem reads

$$u'' = 1, \quad x \in (0,1), \quad u(0) = u(1) = 0\,.$$

'arry out the computations with $N = 0$ and demonstrate that the maximum eflection $u(1/2)$ is the same in the problem utilizing symmetry and the problem overing the whole cable. Filename: `cable_sin.pdf`.

Exercise 25: Check integration by parts

'onsider the Galerkin method for the problem involving u in Exercise 24. Show hat the formulas for c_j are independent of whether we perform integration by arts or not. Filename: `cable_integr_by_parts.pdf`.

Exercise 26: Compute the deflection of a cable with 2 P1 lements

olve the problem for u in Exercise 24 using two P1 linear elements. Filename: `able_2P1.pdf`.

Exercise 27: Compute the deflection of a cable with 1 P2 lement

olve the problem for u in Exercise 24 using one P2 element with quadratic basis unctions. Filename: `cable_1P2.pdf`.

Exercise 28: Compute the deflection of a cable with a step oad

/e consider the deflection of a tension cable as described in Exercise 24. Now he load is

$$\ell(x) = \begin{cases} \ell_1, & x < L/2, \\ \ell_2, & x \geq L/2 \end{cases} \qquad x \in [0, L]\,.$$

his load is not symmetric with respect to the midpoint $x = L/2$ so the solution oses its symmetry and we must solve the scaled problem

$$u'' = \begin{cases} 1, & x < 1/2, \\ 0, & x \geq 1/2 \end{cases} \qquad x \in (0,1), \quad u(0) = 0,\ u(1) = 0\,.$$

) Use $\psi_i = \sin((i+1)\pi x)$, $i = 0, \ldots, N$ and the Galerkin method without ntegration by parts. Derive a formula for c_j in the solution expansion $u = \sum_j c_j\psi_j$. Plot how fast the coefficients c_j tend to zero (on a log scale).

) Solve the problem with P1 finite elements. Plot the solution for $N_e = 2, 4, 8$ ements.
ilename: `cable_discont_load.pdf`.

Exercise 29: Show equivalence between linear systen

Incorporation of Dirichlet conditions at $x = 0$ and $x = L$ in a finite mesh on $\Omega = [0, L]$ can either be done by introducing an expansion $U_0\varphi_0 + U_{N_n}\varphi_{N_n} + \sum_{j=0}^{N} c_j\varphi_{\nu(j)}$, with $N = N_n - 2$ and considering u at the inner nodes as unknowns, *or* one can assemble the matrix syste $u(x) = \sum_{j=0}^{N=N_n} c_j\varphi_j$ and afterwards replace the rows corresponding to k values by the boundary conditions. Show that the two approaches are eq

Exercise 30: Compute with a non-uniform mesh

Derive the linear system for the problem $-u'' = 2$ on $[0, 1]$, with u and $u(1) = 1$, using P1 elements and a *non-uniform* mesh. The vertic coordinates $x_0 = 0 < x_1 < \cdots < x_N = 1$, and the length of cell num $h_e = x_{e+1} - x_e$.

It is of interest to compare the discrete equations for the finite method in a non-uniform mesh with the corresponding discrete equations from a finite difference method. Go through the derivation of the finite di formula $u''(x_i) \approx [D_xD_xu]_i$ and modify it to find a natural discretiz $u''(x_i)$ on a non-uniform mesh. Filename: `nonuniform_P1.pdf`.

Problem 31: Solve a 1D finite element problem by h

The following scaled 1D problem is a very simple, yet relevant, m convective transport in fluids:

$$u' = \epsilon u'', \quad u(0) = 0,\ u(1) = 1,\ x \in [0, 1]\,.$$

a) Find the analytical solution to this problem. (Introduce $w = u'$, s first-order differential equation for $w(x)$, and integrate once more.)

b) Derive the variational form of this problem.

c) Introduce a finite element mesh with uniform partitioning. Use P1 e and compute the element matrix and vector for a general element.

d) Incorporate the boundary conditions and assemble the element contri

e) Identify the resulting linear system as a finite difference discretizatio differential equation using

$$[D_{2x}u = \epsilon D_xD_xu]_i\,.$$

f) Compute the numerical solution and plot it together with the exact for a mesh with 20 elements and $\epsilon = 10, 1, 0.1, 0.01$.
Filename: `convdiff1D_P1.pdf`.

Exercise 32: Compare finite elements and differences for a radially symmetric Poisson equation

We consider the Poisson problem in a disk with radius R with Dirichlet conditions at the boundary. Given that the solution is radially symmetric and hence dependent only on the radial coordinate ($r = \sqrt{x^2 + y^2}$), we can reduce the problem to a 1D Poisson equation

$$-\frac{1}{r}\frac{d}{dr}\left(r\frac{du}{dr}\right) = f(r), \quad r \in (0, R),\ u'(0) = 0,\ u(R) = U_R\,. \tag{310}$$

a) Derive a variational form of (310) by integrating over the whole disk, or equivalently: use a weighting function $2\pi r v(r)$ and integrate r from 0 to R.

b) Use a uniform mesh partition with P1 elements and show what the resulting set of equations becomes. Integrate the matrix entries exact by hand, but use a trapezoidal rule to integrate the f term.

c) Explain that an intuitive finite difference method applied to (310) gives

$$\frac{1}{r_i}\frac{1}{h^2}\left(r_{i+\frac{1}{2}}(u_{i+1} - u_i) - r_{i-\frac{1}{2}}(u_i - u_{i-1})\right) = f_i, \quad i = rh\,.$$

For $i = 0$ the factor $1/r_i$ seemingly becomes problematic. One must always have $u'(0) = 0$, because of the radial symmetry, which implies $u_{-1} = u_1$, if we allow introduction of a fictitious value u_{-1}. Using this u_{-1} in the difference equation for $i = 0$ gives

$$\frac{1}{r_0}\frac{1}{h^2}\left(r_{\frac{1}{2}}(u_1 - u_0) - r_{-\frac{1}{2}}(u_0 - u_1)\right) = \frac{1}{r_0}\frac{1}{2h^2}\left((r_0 + r_1)(u_1 - u_0) - (r_{-1} + r_0)(u_0 - u_1)\right) \approx 2(u_1 - u_0),$$

we use $r_{-1} + r_1 \approx 2r_0$.

Set up the complete set of equations for the finite difference method and compare to the finite element method in case a Trapezoidal rule is used to integrate the f term in the latter method.
Filename: `radial_Poisson1D_P1.pdf`.

Exercise 33: Compute with variable coefficients and P1 elements by hand

Consider the problem

$$-\frac{d}{dx}\left(a(x)\frac{du}{dx}\right) + \gamma u = f(x), \quad x \in \Omega = [0, L], \quad u(0) = \alpha,\ u'(L) = \beta\,. \tag{311}$$

We choose $a(x) = 1 + x^2$. Then

$$u(x) = \alpha + \beta(1 + L^2)\tan^{-1}(x),$$

is an exact solution if $f(x) = \gamma u$.

Derive a variational formulation and compute general expressions element matrix and vector in an arbitrary element, using P1 element uniform partitioning of $[0, L]$. The right-hand side integral is challeng can be computed by a numerical integration rule. The Trapezoidal ru gives particularly simple expressions. Filename: `atan1D_P1.pdf`.

Exercise 34: Solve a 2D Poisson equation using polync and sines

The classical problem of applying a torque to the ends of a rod can be r by a Poisson equation defined in the cross section Ω:

$$-\nabla^2 u = 2, \quad (x, y) \in \Omega,$$

with $u = 0$ on $\partial\Omega$. Exactly the same problem arises for the deflecti membrane with shape Ω under a constant load.

For a circular cross section one can readily find an analytical solutio rectangular cross section the analytical approach ends up with a sine seri idea in this exercise is to use a single basis function to obtain an appr answer.

We assume for simplicity that the cross section is the unit squar $[0, 1] \times [0, 1]$.

a) We consider the basis $\psi_{p,q}(x, y) = \sin((p + 1)\pi x)\sin(q\pi y)$, $p, q =$ These basis functions fulfill the Dirichlet condition. Use a Galerkin met $n = 0$.

b) The basis function involving sine functions are orthogonal. Use th erty in the Galerkin method to derive the coefficients $c_{p,q}$ in a form $\sum_p \sum_q c_{p,q}\psi_{p,q}(x, y)$.

c) Another possible basis is $\psi_i(x, y) = (x(1 - x)y(1 - y))^{i+1}$, $i = 0, \ldots,$ the Galerkin method to compute the solution for $N = 0$. Which cho single basis function is best, $u \sim x(1 - x)y(1 - y)$ or $u \sim \sin(\pi x)\sin($ order to answer the question, it is necessary to search the web or the li for an accurate estimate of the maximum u value at $x = y = 1/2$.
Filename: `torsion_sin_xy.pdf`.

Exercise 35: Analyze a Crank-Nicolson scheme for tl fusion equation

Perform the analysis in Section 19.10 for a 1D diffusion equation u_t discretized by the Crank-Nicolson scheme in time:

$$\frac{u^{n+1}-u^n}{\Delta t} = \alpha\frac{1}{2}\left(\frac{u^{n+1}}{\partial x^2}\frac{u^n}{\partial x^2}\right),$$

r written compactly with finite difference operators,

$$[D_t u = \alpha D_x D_x \overline{u}^t]^{n+\frac{1}{2}}.$$

From a strict mathematical point of view, the u^n and u^{n+1} in these equations ıould be replaced by u_{e}^n and u_{e}^{n+1} to indicate that the unknown is the exact ɔlution of the PDE discretized in time, but not yet in space, see Section 19.1.) Iake plots similar to those in Section 19.10. Filename: `fe_diffusion.pdf`.

References

.] M. G. Larson and F. Bengzon. *The Finite Element Method: Theory, Implementation, and Applications*. Texts in Computational Science and Engineering. Springer, 2013.

Index

affine mapping, 46, 83
approximation
    by sines, 20
    collocation, 24
    interpolation, 25
    of functions, 12
    of general vectors, 9
    of vectors in the plane, 6
assembly, 45

cell, 65
`cells` list, 66
chapeau function, 40
Chebyshev nodes, 29
collocation method (approximation), 24

degree of freedom, 65
dof map, 65
`dof_map` list, 66

edges, 81
element matrix, 44
essential boundary condition, 109

faces, 81
finite element basis function, 40
finite element expansion
    reference element, 65
finite element mesh, 34
finite element, definition, 65

Galerkin method
    functions, 13
    vectors, 8, 11
Gauss-Legendre quadrature, 71

hat function, 40
Hermite polynomials, 69

integration by parts, 102
interpolation, 25
isoparametric mapping, 83

Kronecker delta, 27, 36

Lagrange (interpolating) poly
    26
least squreas method
    vectors, 7
linear elements, 40
lumped mass matrix, 64, 148

mapping of reference cells
    affine mapping, 46
    isoparametric mapping, 8
mass lumping, 64, 148
mass matrix, 64, 146, 148
mesh
    finite elements, 34
Midpoint rule, 70
mixed finite elements, 164

natural boundary condition, 1
Newton-Cotes rules, 70
numerical integration
    Midpoint rule, 70
    Newton-Cotes formulas, 7
    Simpson's rule, 70
    Trapezoidal rule, 70

P1 element, 40
P2 element, 40
projection
    functions, 13
    vectors, 8, 11

quadratic elements, 40

reference cell, 65
residual, 95
Runge's phenomenon, 29

simplex elements, 81
simplices, 81
Simpson's rule, 70
sparse matrices, 58
stiffness matrix, 146
strong form, 103

ensor product, 72
est function, 97
est space, 97
rapezoidal rule, 70
rial function, 97
rial space, 97

ariational formulation, 96
ertex, 65
`ertices` list, 66

eak form, 103